Supplementary Analysis

"Single-cell reconstruction of developmental trajectories during zebrafish embryogenesis"

Jeffrey A. Farrell*, Yiqun Wang*, Samantha J. Riesenfeld, Karthik Shekhar, Aviv Regev†, Alexander F. Schier†

Science 26 Apr 2018. doi: 10.1126/science.aar3131

URD 1: Creating URD Object and Finding Variable Genes

```
library(URD)
library(gridExtra)  # grid.arrange function
```

Load filtered data

We load `zf.dropseq.counts`, which is a *genes X cells* data.frame of unnormalized, unlogged transcripts detected per gene per cell. We also load `zf.dropseq.meta`, which is a *cells X metadata* data.frame of metadata about each cell (*e.g.* number of genes detected, number of cells detected, sequencing batch, developmental stage, and so on.).

```
zf.dropseq.counts <- readRDS(file = "data/zf.dropseq.counts.rds")
zf.dropseq.meta <- readRDS(file = "data/zf.dropseq.meta.rds")
```

Create an URD object

```
# Create URD object
object <- createURD(count.data = zf.dropseq.counts, meta = zf.dropseq.meta, min.cells = 20,
    min.counts = 20, gene.max.cut = 5000)
```

```
## 2018-02-08 20:02:37: Filtering cells by number of genes.
## 2018-02-08 20:03:44: Filtering genes by number of cells.
## 2018-02-08 20:05:08: Filtering genes by number of counts across entire data.
## 2018-02-08 20:06:38: Filtering genes by maximum observed expression.
## 2018-02-08 20:08:06: Creating URD object.
## 2018-02-08 20:08:10: Determining normalization factors.
## 2018-02-08 20:08:49: Normalizing and log-transforming the data.
## 2018-02-08 20:10:22: Finishing setup of the URD object.
## 2018-02-08 20:11:00: All done.
```

```
# Delete the original data
rm(list = c("zf.dropseq.counts", "zf.dropseq.meta"))

# Perform garbage collection to free RAM.
shhhh <- gc()
```

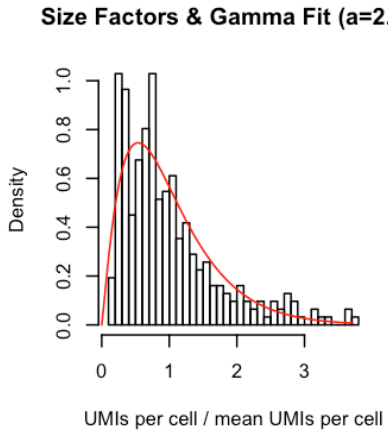
Find variable genes

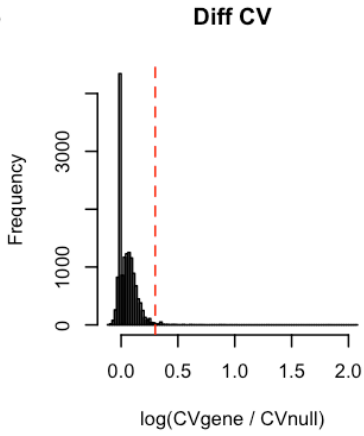
Because scRNA-seq data is noisy, gene expression exhibits high variability due to technical effects, and the amount of technical variability is linked to mean expression level in scRNA-seq data. To identify genes that are likely to encode biologically relevant information, we look for those that exhibit more variability than other similarly expressed genes. We use only those highly variable genes for calculating distance between cells in gene expression space, for calculating the diffusion map, for building the tree, and we also privilege them during differential expression with lower thresholds (as they are more likely to be interesting cell-type specific markers).

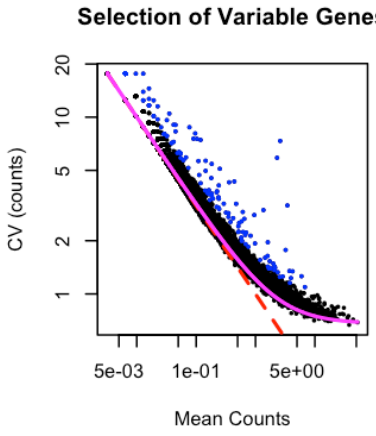
As the genes that encode biological information may change over developmental time, we calculate variable genes separately for each stage, and then take the union of them.

```
# Find a list of cells from each stage.
stages <- unique(object@meta$STAGE)
cells.each.stage <- lapply(stages, function(stage) rownames(object@meta)[which(object@meta$STAGE ==
    stage)])

# Compute variable genes for each stage.
var.genes.by.stage <- lapply(1:length(stages), function(n) findVariableGenes(object,
    cells.fit = cells.each.stage[[n]], set.object.var.genes = F, diffCV.cutoff = 0.3,
    mean.min = 0.005, mean.max = 100, main.use = stages[[n]], do.plot = T))
```

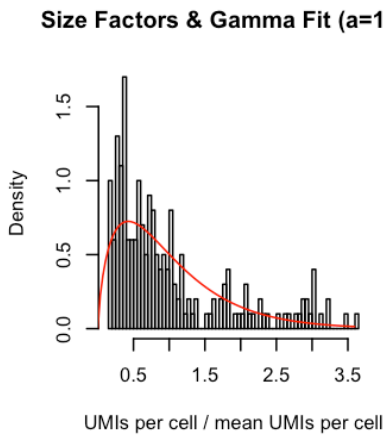
ZFHIGH

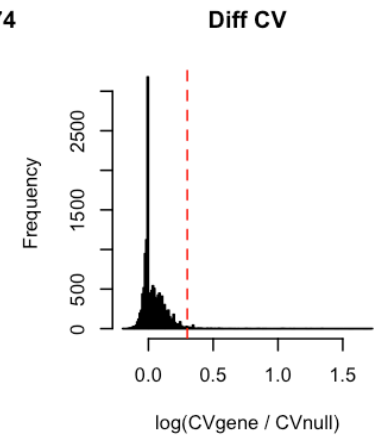


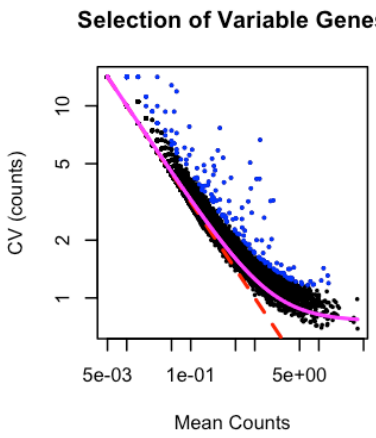




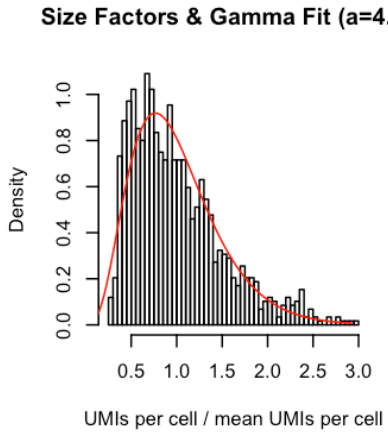
ZFOBLONG

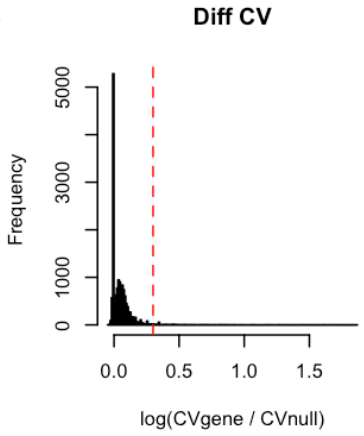


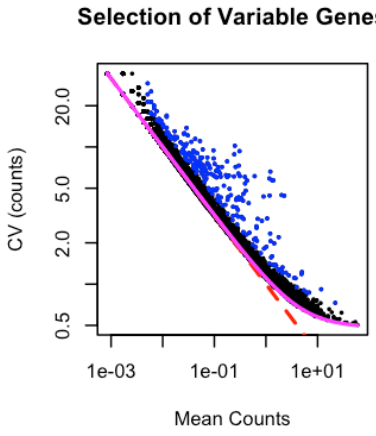




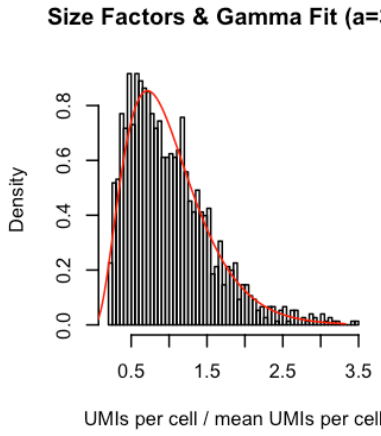
ZFDOME

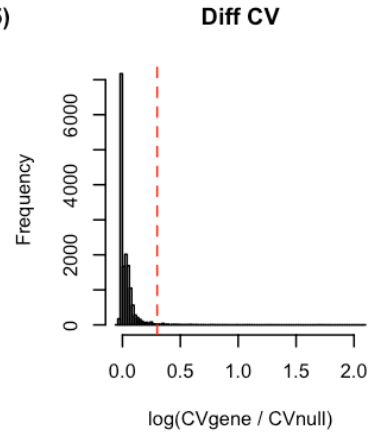


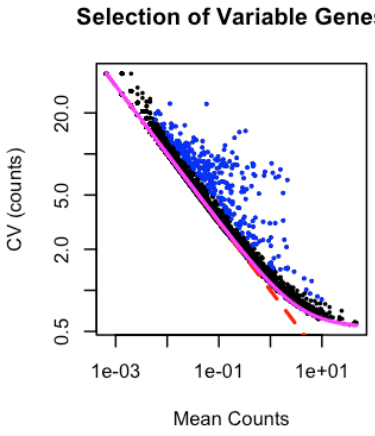




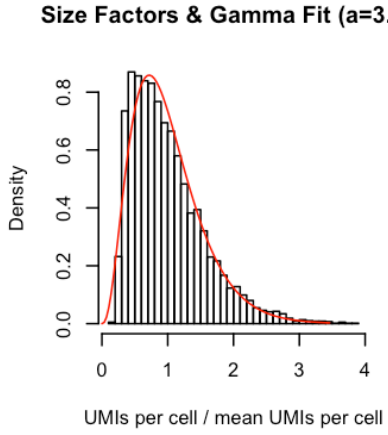
ZF30

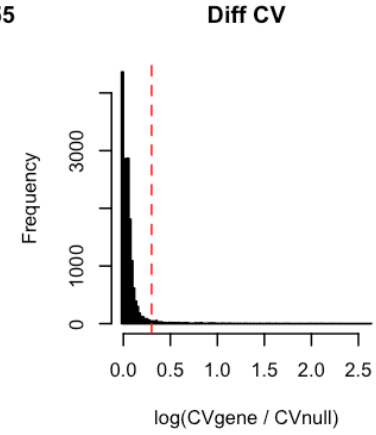


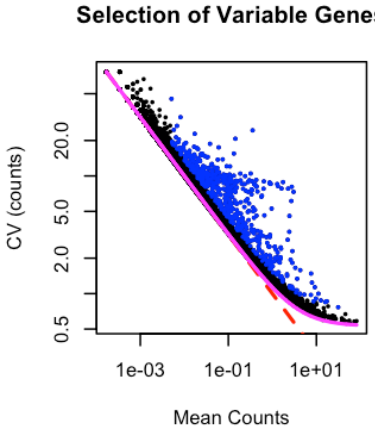




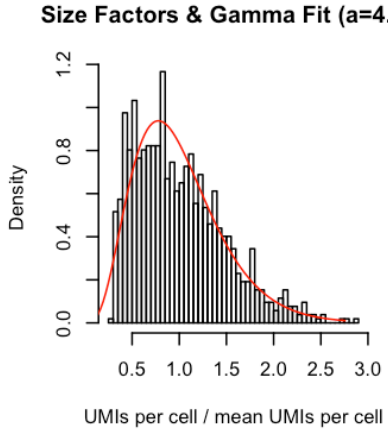
ZF50

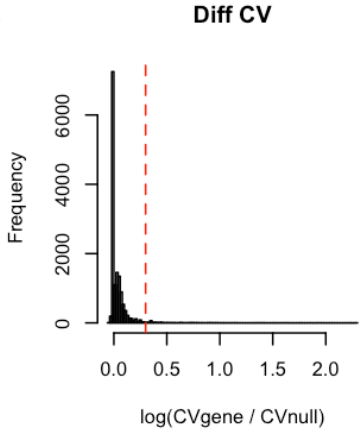


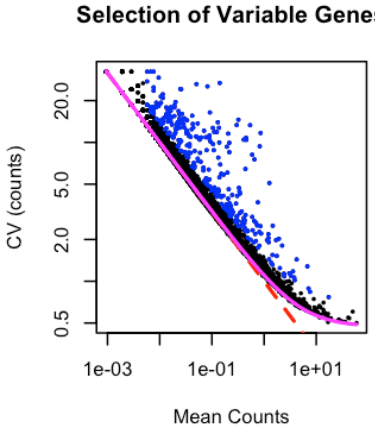




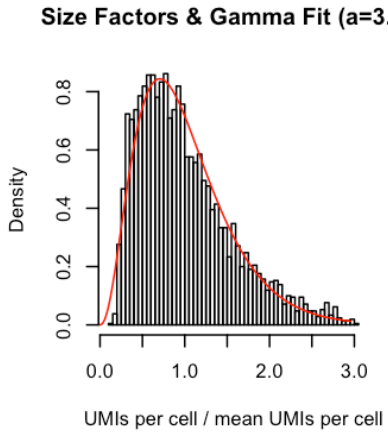
ZFS

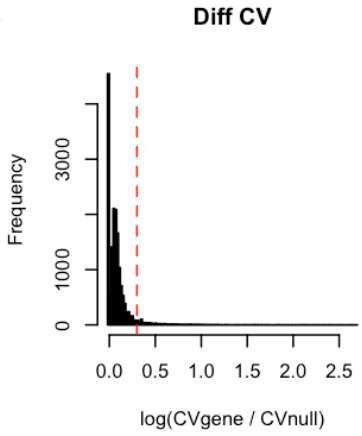


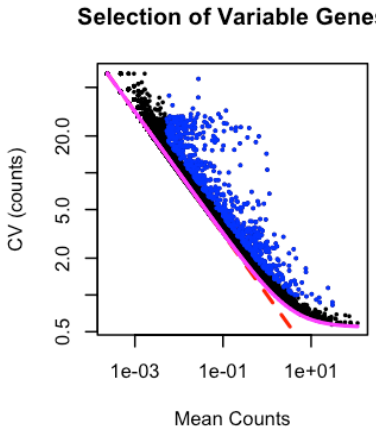




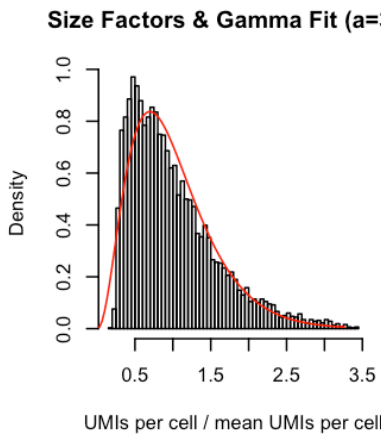
ZF60

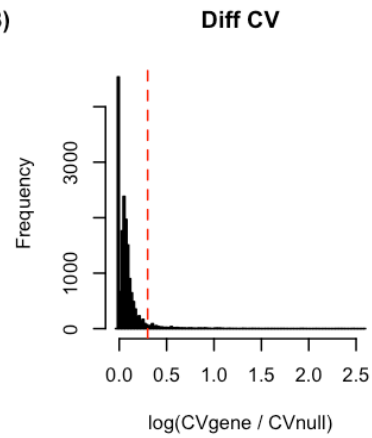


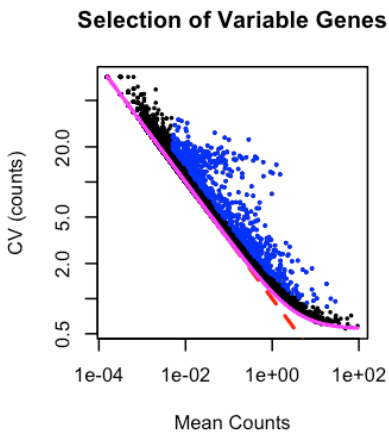




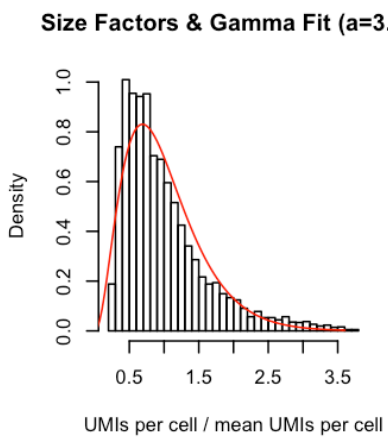
ZF75

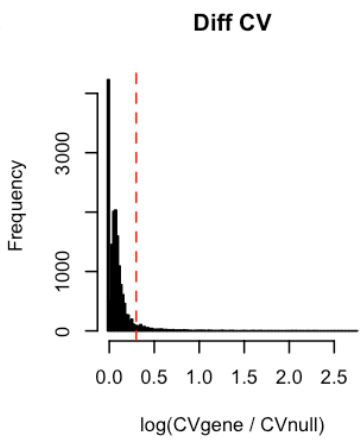


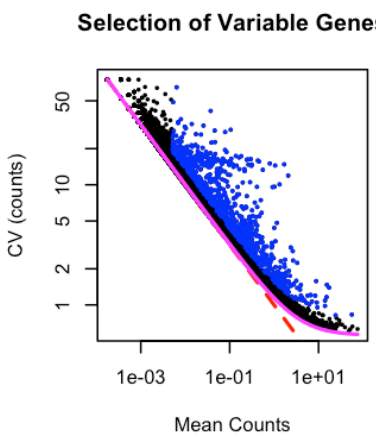


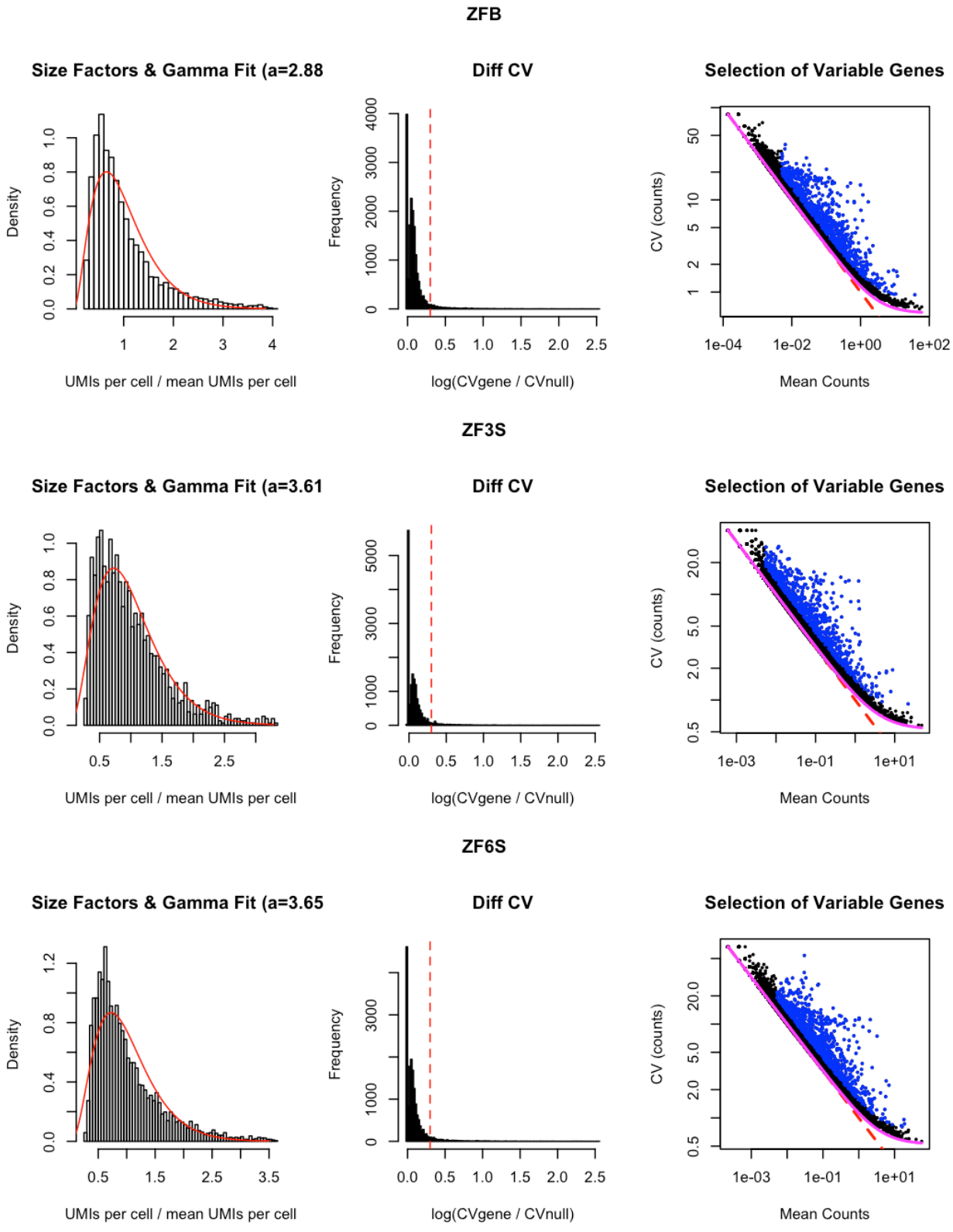


ZF90










```
names(var.genes.by.stage) <- stages

# Take union of variable genes from all stages
var.genes <- sort(unique(unlist(var.genes.by.stage)))
```

```
# Set variable genes in object
object@var.genes <- var.genes

# Save variable gene lists
for (stage in stages) {
    write(var.genes.by.stage[[stage]], file = paste0("var_genes/var_", stage, ".txt"))
}
write(var.genes, file = "var_genes/var_genes.txt")
```

Perform PCA and calculate a tSNE projection

These steps are not strictly necessary for building a tree using URD, but they are a common visualization and can be useful for inspecting the data. The PCA is also required for graph-based clustering which we use below to remove some populations that would confound the discovery of developmental trajectories.

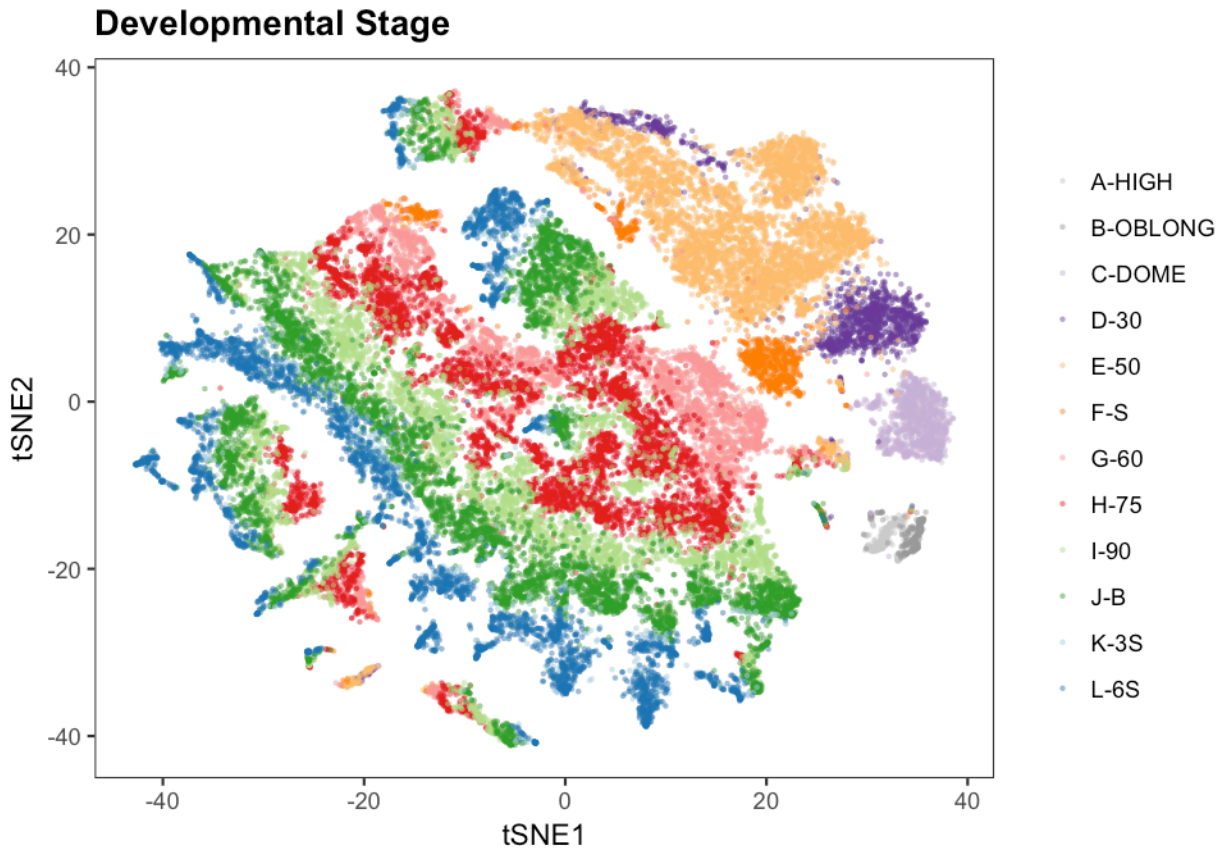
```
object <- calcPCA(object)
```

```
## [1] "2018-02-08 20:12:34: Centering and scaling data."
## [1] "2018-02-08 20:12:57: Removing genes with no variation."
## [1] "2018-02-08 20:13:05: Calculating PCA."
## [1] "2018-02-08 20:21:10: Estimating significant PCs."
## [1] "Marchenko-Pastur eigenvalue null upper bound: 1.48442498751452"
## [1] "97 PCs have larger eigenvalues."
## [1] "Storing 194 PCs."
```

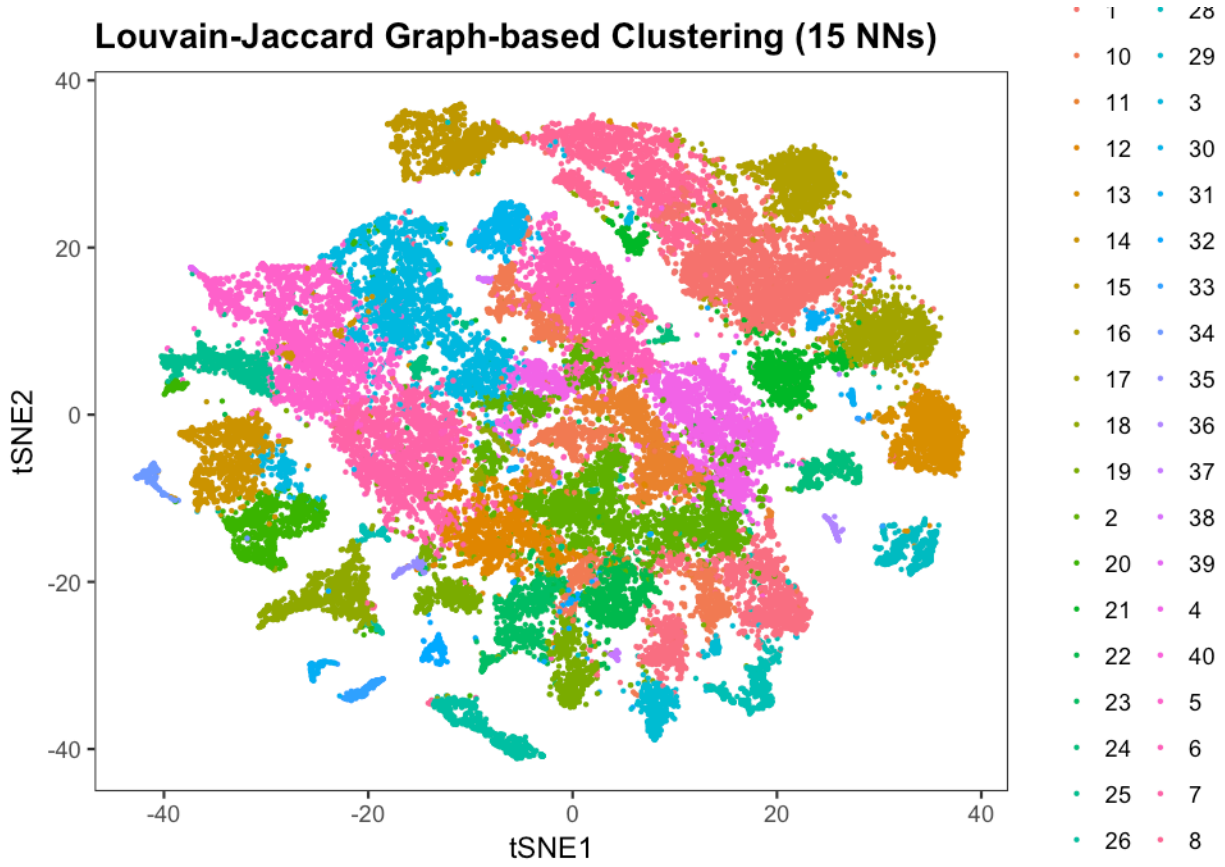
```
set.seed(18)
object <- calcTsne(object, perplexity = 30, theta = 0.5)
```

```
set.seed(17)
object <- graphClustering(object, dim.use = "pca", num.nn = c(15, 20, 30), do.jaccard = T,
    method = "Louvain")
```

```
# Need to make a new version of stage names that is alphabetical.
object@meta$stage.nice <- plyr::mapvalues(x = object@meta$STAGE, from = c("ZFHIGH",
    "ZFOBLONG", "ZFDOME", "ZF30", "ZF50", "ZFS", "ZF60", "ZF75", "ZF90", "ZFB", "ZF3S",
    "ZF6S"), to = c("A-HIGH", "B-OBLONG", "C-DOME", "D-30", "E-50", "F-S", "G-60",
    "H-75", "I-90", "J-B", "K-3S", "L-6S"))
stage.colors <- c("#CCCCCC", RColorBrewer::brewer.pal(9, "Set1")[9], RColorBrewer::brewer.pal(12,
    "Paired")[c(9, 10, 7, 8, 5, 6, 3, 4, 1, 2)])

plotDim(object, "stage.nice", discrete.colors = stage.colors, legend = T, plot.title = "Developmental Stage",
    alpha = 0.5)
```




```
plotDim(object, "Louvain-15", legend = T, plot.title = "Louvain-Jaccard Graph-based Clustering (15 NNs)",
    alpha = 1)
```



Remove outliers

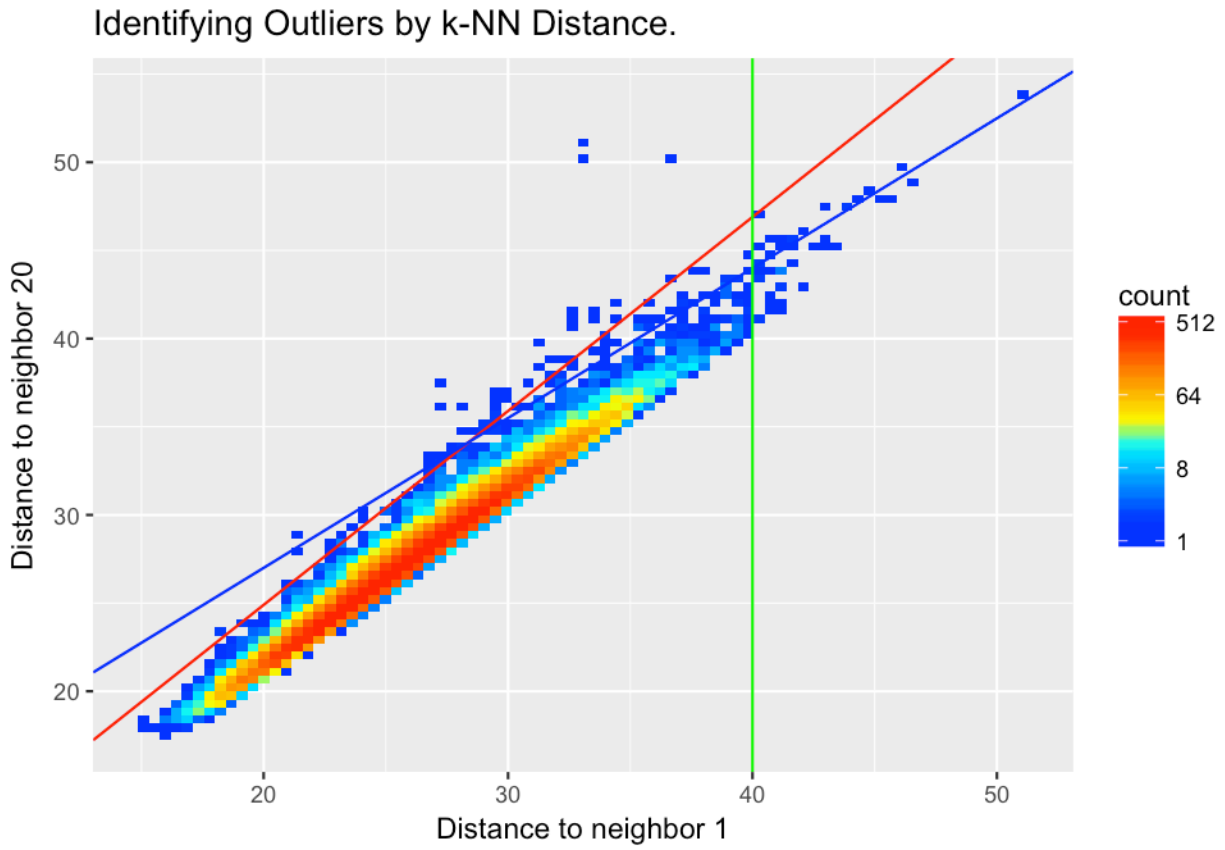
Identify cells that are poorly connected

Since the diffusion map is calculated on a k-nearest neighbor graph in gene expression space, cells that are unusually far from their nearest neighbors in a k-nearest neighbor graph often result in poor diffusion maps because many of the highly ranked diffusion components will primarily represent variability of individual outlier cells. Thus, cropping cells based on their distance to their nearest neighbor, and cropping cells that have unusually large distances to an nth nearest neighbor (given the distance to their nearest neighbor) generally produces better, more connected diffusion maps.

```
# Calculate a k-nearest neighbor graph
object <- calcKNN(object, nn = 100)
```

We cropped cells to the right of the green line (those that are unusually far from their nearest neighbor) and cells above the blue or red lines (those that are unusually far from their 20th nearest neighbor, given their distance to their 1st nearest neighbor).

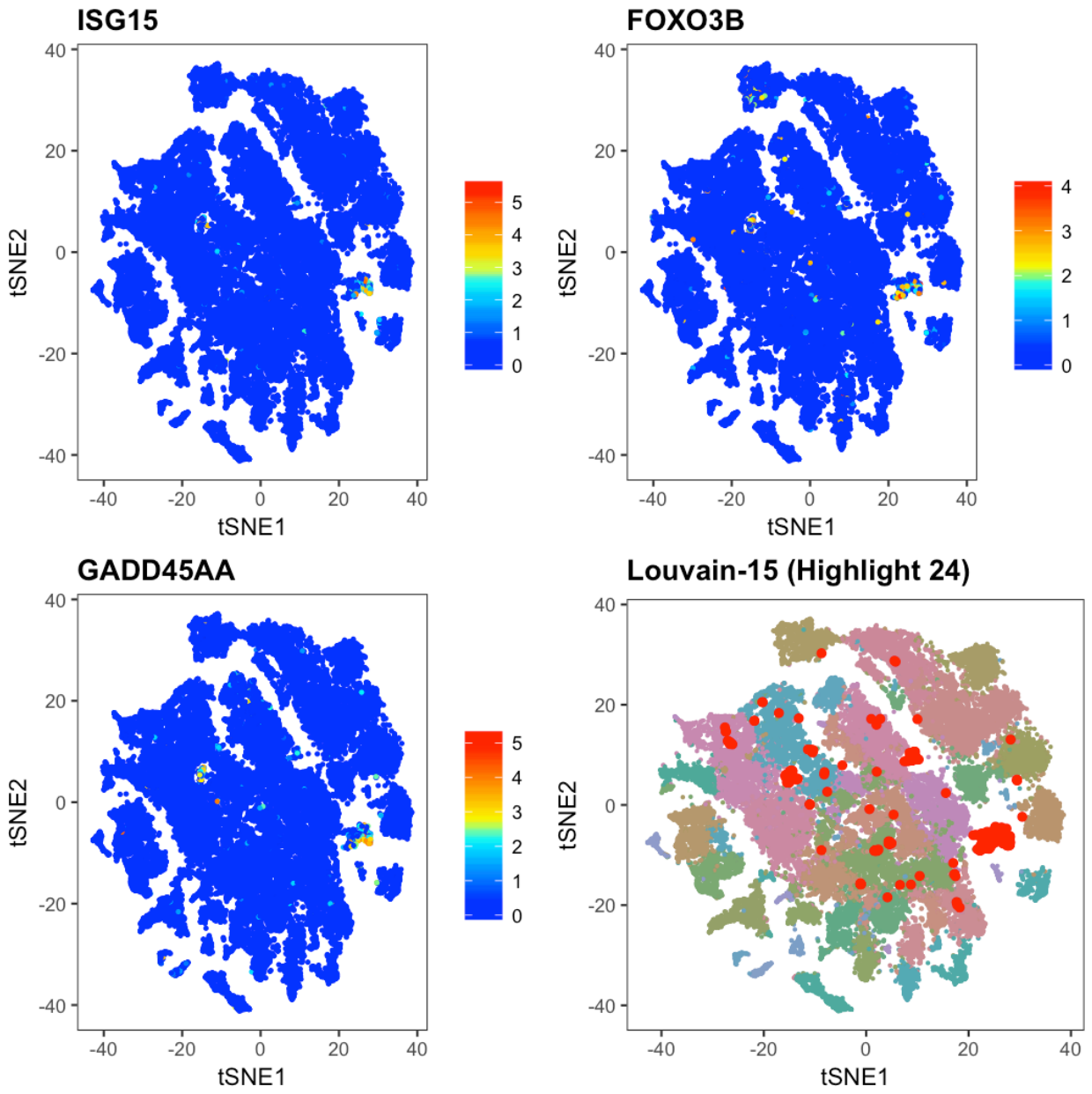
```
# Plot cells according to their distance to their nearest and 20th nearest
# neighbors, and identify those with unusually large distances.
outliers <- knnOutliers(object, nn.1 = 1, nn.2 = 20, x.max = 40, slope.r = 1.1, int.r = 2.9,
    slope.b = 0.85, int.b = 10, title = "Identifying Outliers by k-NN Distance.")
```



Identify apoptotic-like cells

A group of cells had very strong expression for the 'apoptotic-like' program that was identified in our prior work (Satija and Farrell, Gennert, Schier and Regev; Nature Biotechnology 2015). These cells seem to arise from many different cell types, and express both a cell-type specific program, as well as the 'apoptotic-like' program. We removed them from further analysis, because we reasoned that their shared state would create 'short-circuits' in the developmental trajectories; cells from already-distinct cell types could be connected in gene expression space through their common expression of this strong cell-type-independent program. These cells primarily occupied a single cluster in Louvain-Jaccard clustering on the entire data set (identified through expression of this cell state's markers isg15, foxo3b, and gadd45aa). Cells in this cluster were removed from further analysis.

```
gridExtra::grid.arrange(grobs=list(
  # Plot some apoptotic-like markers
  plotDim(object, "ISG15", alpha=0.4, point.size=0.5),
  plotDim(object, "FOXO3B", alpha=0.4, point.size=0.5),
  plotDim(object, "GADD45AA", alpha=0.4, point.size=0.5),
  # Figure out which cluster corresponds to these cells
  plotDimHighlight(object, clustering="Louvain-15", cluster="24", legend=F)
))
```




```
apoptotic.like.cells <- cellsInCluster(object, "Louvain-15", "24")
```

Subset object to eliminate outliers

```
cells.keep <- setdiff(colnames(object@logupx.data), c(outliers, apoptotic.like.cells))
object <- urdSubset(object, cells.keep = cells.keep)
```

Save object

```
saveRDS(object, file = "obj/object_2_trimmed.rds")
```

URD 2: Diffusion Map and Pseudotime

```
library(URD)
```

Load previous saved object

```
object <- readRDS("obj/object_2_trimmed.rds")
```

Calculate diffusion map

First, from the data, we have to calculate the transition probalities between cells. The eigendecomposition of the transition probabilities gives diffusion components (which comprise a diffusion map). Inspecting the diffusion map is an easy way to verify that good parameters have been chosen for the transition probabilities. For this, we use the pioneering package *destiny* from the Theis lab that established the usefulness of diffusion maps for studying differentiation processes in single-cell RNAseq data.

Run on the cluster

For smaller data sets (e.g. 10,000 cells), the diffusion map can easily be calculated on your laptop or desktop computer. For larger data sets, such as the one here (~40,000 cells), it can be RAM intensive, so diffusion maps were calculated on the cluster using the scripts URD-DM.R and URD-DM.sh. The commands included in URD-DM.R were:

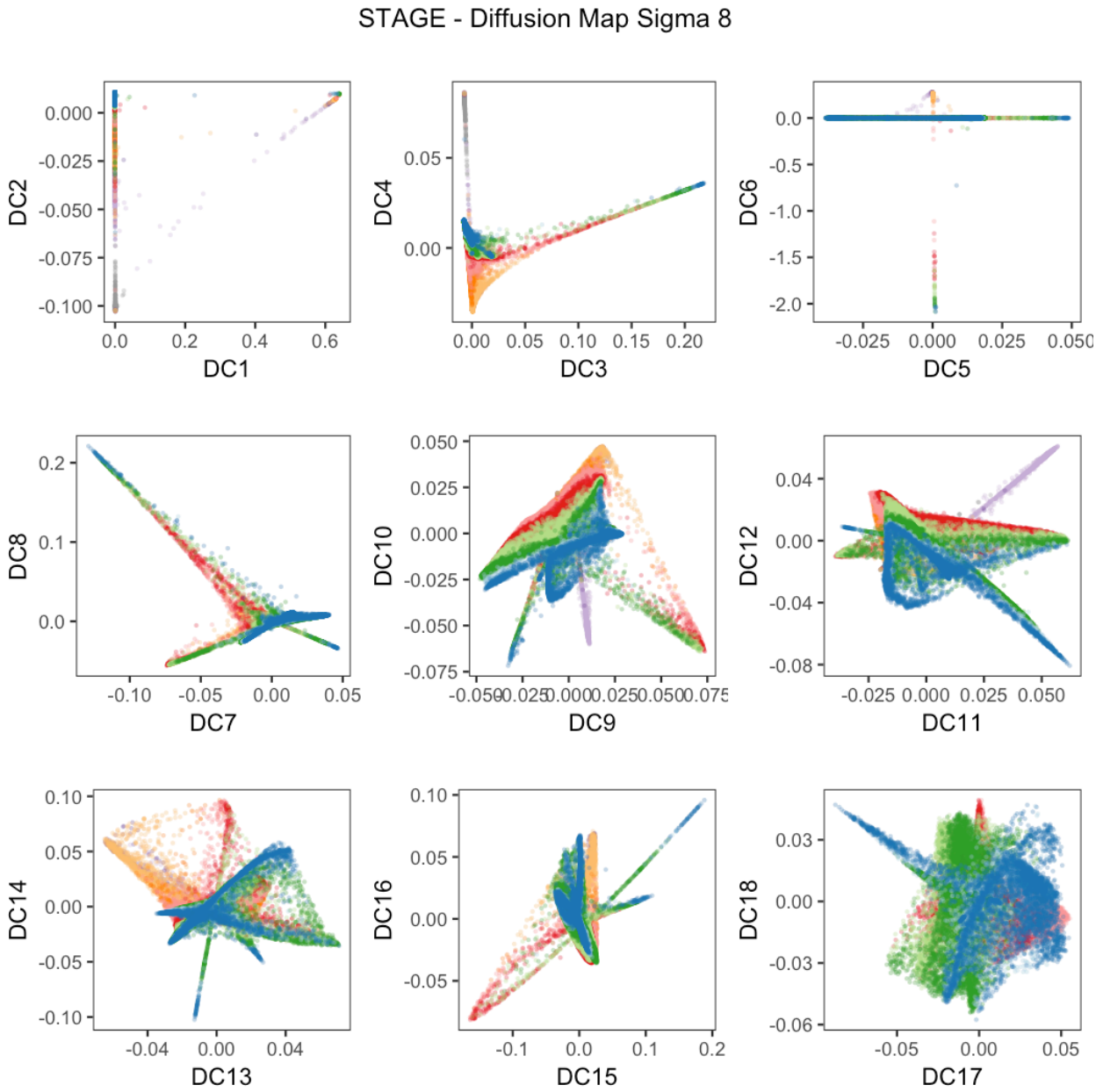
```
# Calculate diffusion map
object <- calcDM(object, knn = 200, sigma.use = 8)

# Save diffusion map to read in later.
saveRDS(object@dm, file = "dm/dm-8-2.0.6ep.rds")
```

Load diffusion map and add to URD object

```
# Load calculated diffusion map
dm.8 <- readRDS("dm/dm-8-2.0.6ep.rds")

# Add it to the URD object
object <- importDM(object, dm.8)
```

Inspect diffusion map

```
# Stage color palette
stage.colors <- c("#CCCCCC", RColorBrewer::brewer.pal(9, "Set1")[9], RColorBrewer::brewer.pal(12,
    "Paired")[c(9, 10, 7, 8, 5, 6, 3, 4, 1, 2)])

plotDimArray(object = object, reduction.use = "dm", dims.to.plot = 1:18, label = "stage.nice",
    plot.title = "", outer.title = "STAGE - Diffusion Map Sigma 8", legend = F, alpha = 0.3,
    discrete.colors = stage.colors)
```



We have had good results tuning sigma such that 1 or 2 pairs of DCs become very tight, while the remainder exhibit sharp spikes for several more DCs before beginning to become blurry. For this data, sigma 8 exhibits that desired behavior. (Additional diffusion maps with varied sigma values are presented in "URD: Choosing Parameters - Diffusion Map Sigma.")

Pseudotime

We next assign cells a pseudotime, that represents an ordering in the process of differentiation. We find that cells from neighboring developmental stages can exhibit extremely similar transcriptomes, so we prefer this to analyzing the data according to its developmental stage. Pseudotime is used later for biasing random walks used to determine developmental trajectories, as well as for determining where branchpoints lie in the data.

Calculate pseudotime "floods"

Because this computation can take some time and RAM, this portion was also run on the cluster, using the script URD-PT.R and URD-PT.sh. Since many simulations are run, this allows the work to be split across several CPUs. For smaller datasets (<10,000 cells), this can be efficiently run on a laptop. The commands run in URD-PT.R were:

We first define the 'root' or the base of the specification tree as a group of cells (here, all cells from the first developmental stage we profiled). We then simulate 'floods', which start with the root cells visited, and move to connected cells in a stepwise fashion (with the chance of visiting a neighboring cell determined by the transition probabilities). This continues until the visitation structure of the graph stops changing much in a given iteration.

```
# Define the root cells as cells in HIGH stage
root.cells <- rownames(object@meta)[object@meta$STAGE == "ZFHIGH"]

# Do the flood
flood.result <- floodPseudotime(object, root.cells = root.cells, n = 10, minimum.cells.flooded = 2,
    verbose = T)

# Save the result
saveRDS(flood.result, file = "floods/flood-dm-8-[random#].rds")
```

Process pseudotime floods

We loaded the pre-run floods from the cluster (where 150 total simulations were performed).

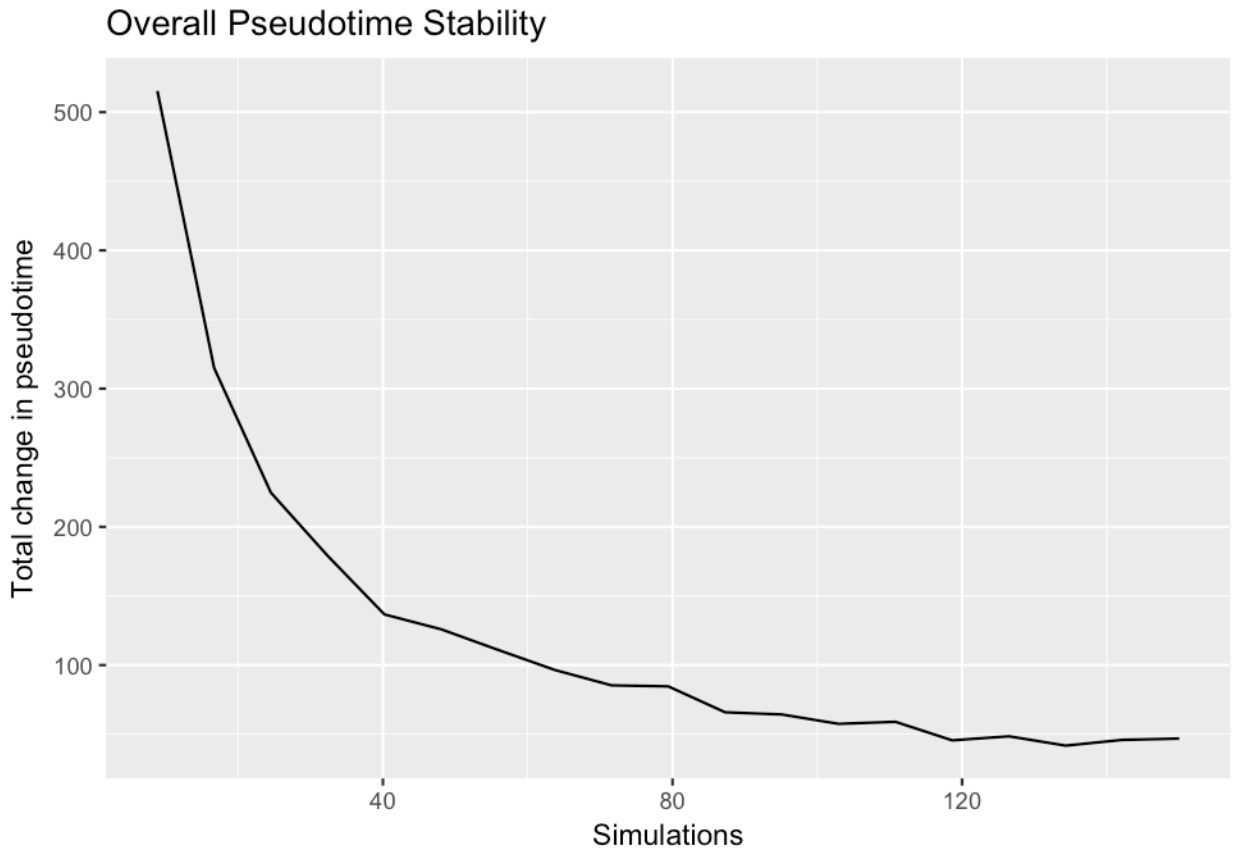
```
# Load floods
floods.dm8 <- lapply(list.files(path = "floods/", pattern = "flood-dm-8-", full.names = T),
    readRDS)
```

We then processed the simulations. Each cell was assigned pseudotime determined as the average across all simulations of the step that visited the cell (normalized to the number of steps in a given simulation).

```
# Process the floods
object <- floodPseudotimeProcess(object, floods.dm8, floods.name = "pseudotime",
    max.frac.NA = 0.4, pseudotime.fun = mean, stability.div = 20)
```

Pseudotime was calculated with several sub-sampled portions of the simulations, and the overall change in pseudotime across all cells was determined as more data was added. Since this graph reaches an asymptote, enough simulations were performed.

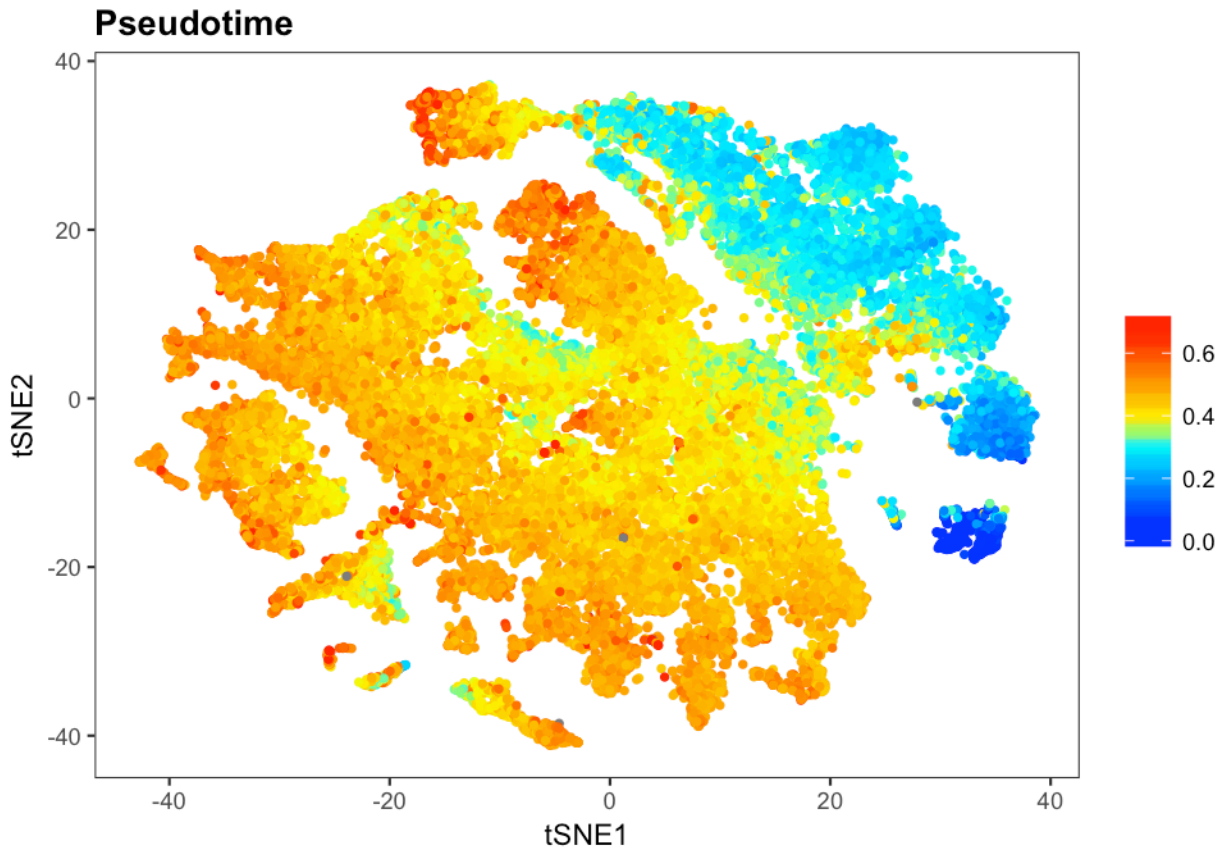
```
pseudotimePlotStabilityOverall(object)
```



Inspect pseudotime

The detected pseudotime looked like this, shown on the tSNE.

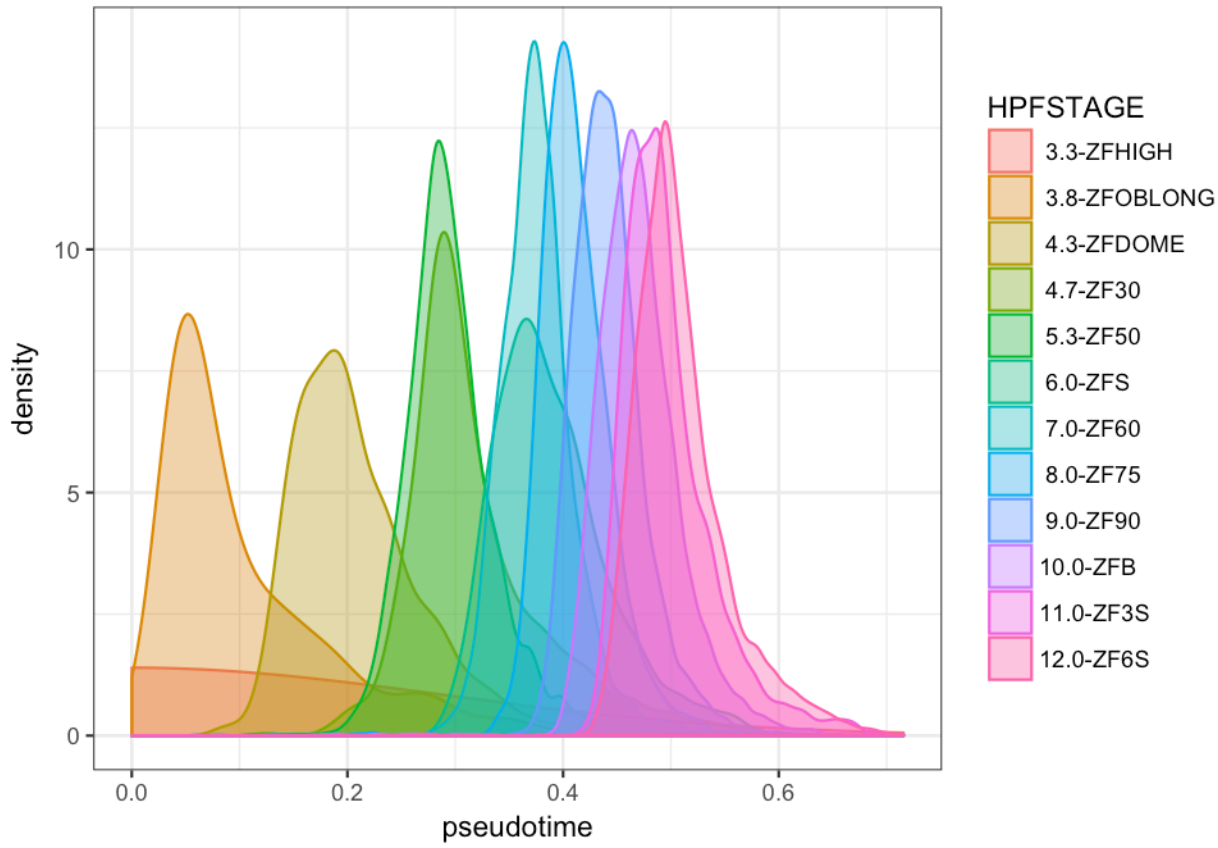
```
plotDim(object, "pseudotime", plot.title = "Pseudotime")
```



And we the plotted the distribution of pseudotime for cells from each developmental stage. As expected, there is a clear correlation between pseudotime and actual developmental stage, but the distributions of pseudotime overlap for neighboring stages, which is in accord with our expectations of developmental asynchrony. All cells from 3.3 HPF - ZFHIGH have pseudotime 0 (because they were defined as the root), which creates an odd shape in this density plot.

```
# Define a properly ordered stage name.
object@meta$HPFSTAGE <- apply(object@meta[, c("HPF", "STAGE")], 1, paste0, collapse = "-",
    sep = "")

# Create a data.frame that includes pseudotime and stage information
gg.data <- cbind(object@pseudotime, object@meta[rownames(object@pseudotime), ])

# Plot
ggplot(gg.data, aes(x = pseudotime, color = HPFSTAGE, fill = HPFSTAGE)) + geom_density(alpha = 0.4) +
    theme_bw()
```

```
## Warning: Removed 19 rows containing non-finite values (stat_density).
```



Save object

```
saveRDS(object, file = "obj/object_3_withDMandPT.rds")
```

URD 3: Determining Tips

```
library(URD)
library(scran)  # Batch correction using MNN
library(gridExtra)  # grid.arrange
```

Load previous saved object

```
object <- readRDS("obj/object_3_withDMandPT.rds")
```

Trim to cells from final stage

URD requires users to define the 'tips' of the developmental tree (i.e. the terminal populations). To do this, we clustered the data from the final stage of our timecourse, and decided which clusters to use as tips.

The first step is to trim the data to the cells from the final stage.

```
# Subset the object
cells.6s <- grep("ZF6S", colnames(object@logupx.data), value = T)
object.6s <- urdSubset(object, cells.keep = cells.6s)
```

Perform PCA / tSNE on final stage cells

Then, we loaded the variable genes specific to this stage from our earlier calculations (see Part 1), and performed PCA, graph-based clustering, and tSNE (to easily visualize the clustering).

```
# Load the variable genes specific to this stage
var.genes.6s <- scan("var_genes/var_ZF6S.txt", what = "character")
object.6s@var.genes <- var.genes.6s

# Calculate PCA
object.6s <- calcPCA(object.6s)
```

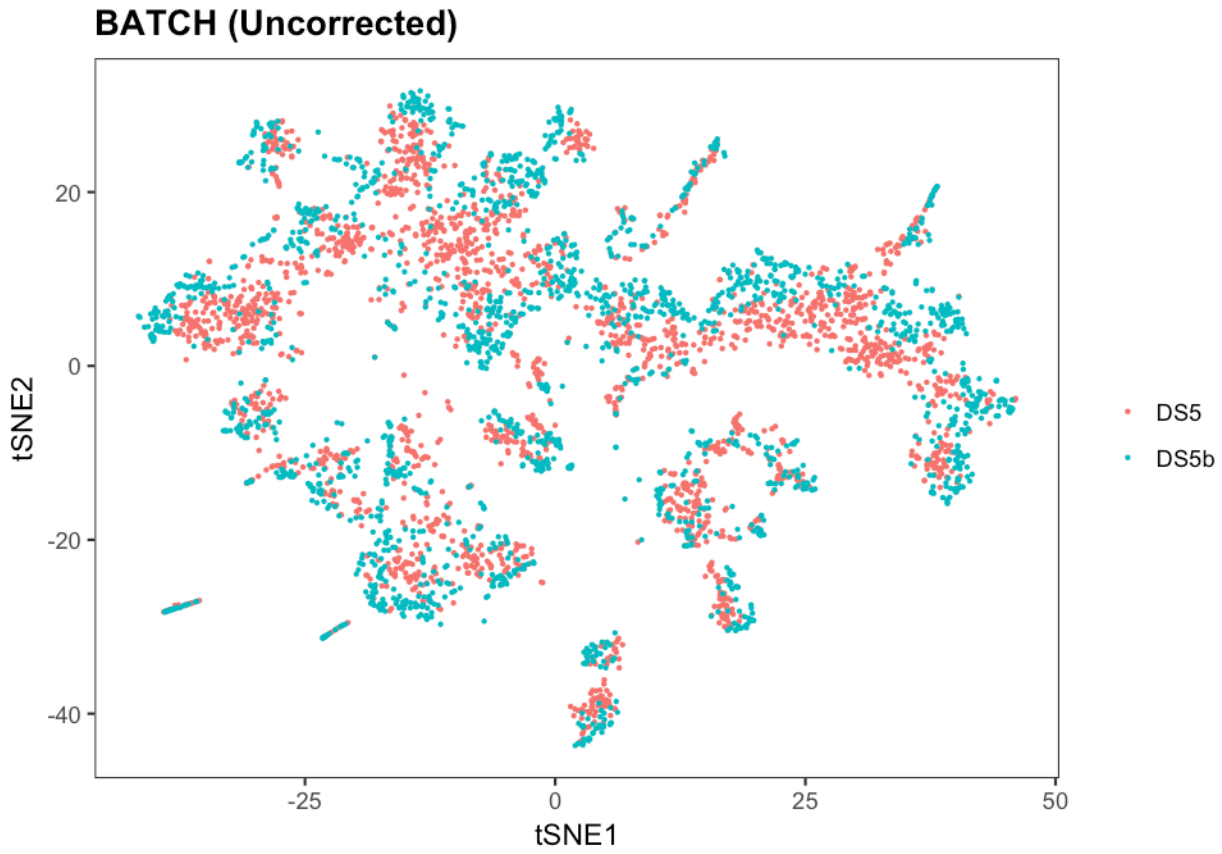
```
## Warning in as.POSIXlt.POSIXct(x, tz): unknown timezone 'zone/tz/2017c.1.0/
## zoneinfo/America/New_York'
```

```
## [1] "2018-02-09 03:00:34: Centering and scaling data."
## [1] "2018-02-09 03:00:35: Removing genes with no variation."
## [1] "2018-02-09 03:00:36: Calculating PCA."
## [1] "2018-02-09 03:00:49: Estimating significant PCs."
## [1] "Marchenko-Pastur eigenvalue null upper bound: 2.11430880049087"
## [1] "45 PCs have larger eigenvalues."
## [1] "Storing 90 PCs."
```

```
# Calculate tSNE
set.seed(18)
object.6s <- calcTsne(object.6s, dim.use = "pca", perplexity = 30, theta = 0.5)
```

We noticed that, in this context, there was a noticeable batch effect between our two samples, and it sometimes drove cluster boundaries in the data, which was not desired.

```
# Look at batch information
plotDim(object.6s, "BATCH", plot.title = "BATCH (Uncorrected)")
```



Batch correct data

Therefore, we batch-corrected the data. We used MNN (https://doi.org/10.1101/165118), a single-cell aware batch correction algorithm that finds mutually nearest neighbors between batches, and uses them to calculate correction vectors.

```
# Make a copy of the object
object.6s.mnn <- object.6s

# Generate expression matrices from each batch
cells.1 <- rownames(object.6s.mnn@meta)[which(object.6s.mnn@meta$BATCH == "DS5")]
cells.2 <- rownames(object.6s.mnn@meta)[which(object.6s.mnn@meta$BATCH == "DS5b")]
exp.1 <- as.matrix(object.6s.mnn@logupx.data[, cells.1])
exp.2 <- as.matrix(object.6s.mnn@logupx.data[, cells.2])

# Batch correct using MNN, correcting all genes, but using the variable genes to
# determine mutual nearest neighbors.
logupx.6s.mnn <- mnnCorrect(exp.1, exp.2, subset.row = object.6s.mnn@var.genes, k = 20,
    sigma = 1, svd.dim = 0, cos.norm.in = T, cos.norm.out = F)

# Combine the resultant corrected matrices and return to original order of cells
logupx.6s.mnn <- do.call("cbind", logupx.6s.mnn[[1]])
logupx.6s.mnn <- logupx.6s.mnn[, colnames(object.6s.mnn@logupx.data)]

# Re-sparsify the matrix, by turning anything less than 0 or near 0 back to 0.
logupx.6s.mnn[logupx.6s.mnn < 0.05] <- 0
object.6s.mnn@logupx.data <- as(logupx.6s.mnn, "dgCMatrix")

# Re-calculate PCA
object.6s.mnn <- calcPCA(object.6s.mnn)
```

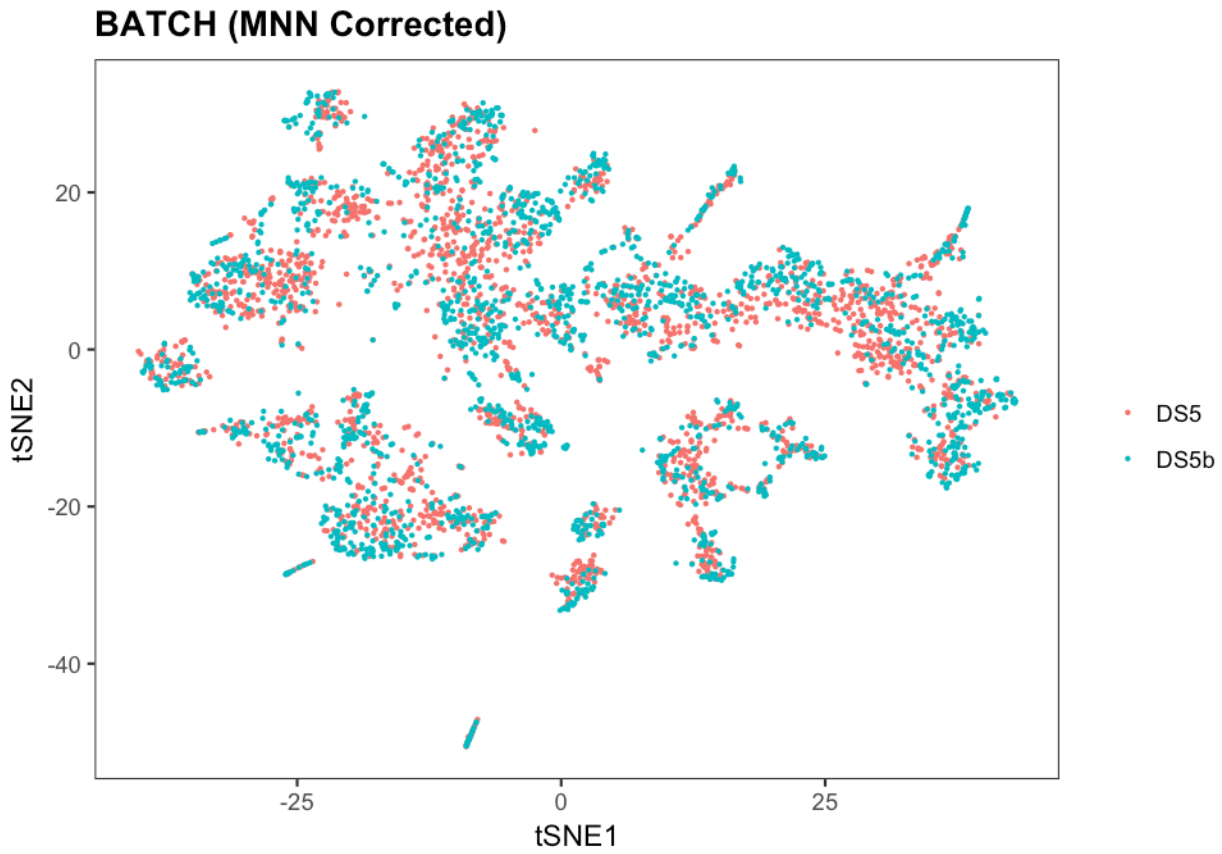
```
## [1] "2018-02-09 03:03:57: Centering and scaling data."
## [1] "2018-02-09 03:03:59: Removing genes with no variation."
## [1] "2018-02-09 03:03:59: Calculating PCA."
```

```
## [1] "2018-02-09 03:04:12: Estimating significant PCs."
## [1] "Marchenko-Pastur eigenvalue null upper bound: 2.11430880049087"
## [1] "45 PCs have larger eigenvalues."
## [1] "Storing 90 PCs."
```

```
# Re-calculate tSNE
set.seed(18)
object.6s.mnn <- calcTsne(object.6s.mnn, dim.use = "pca", perplexity = 30, theta = 0.5)
```

We found that this ameliorated the batch effect in the data.

```
# Look at batch information
plotDim(object.6s.mnn, "BATCH", plot.title = "BATCH (MNN Corrected)")
```



Cluster cells from final stage

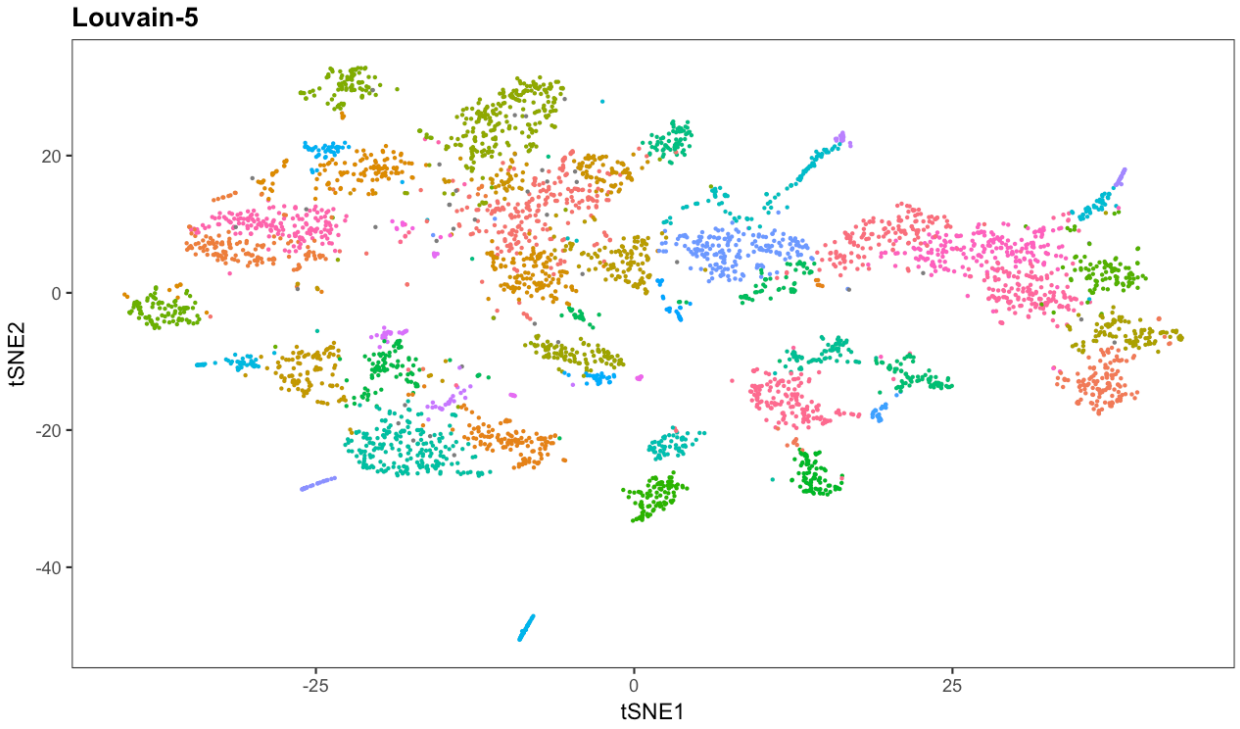
Do graph-based clustering

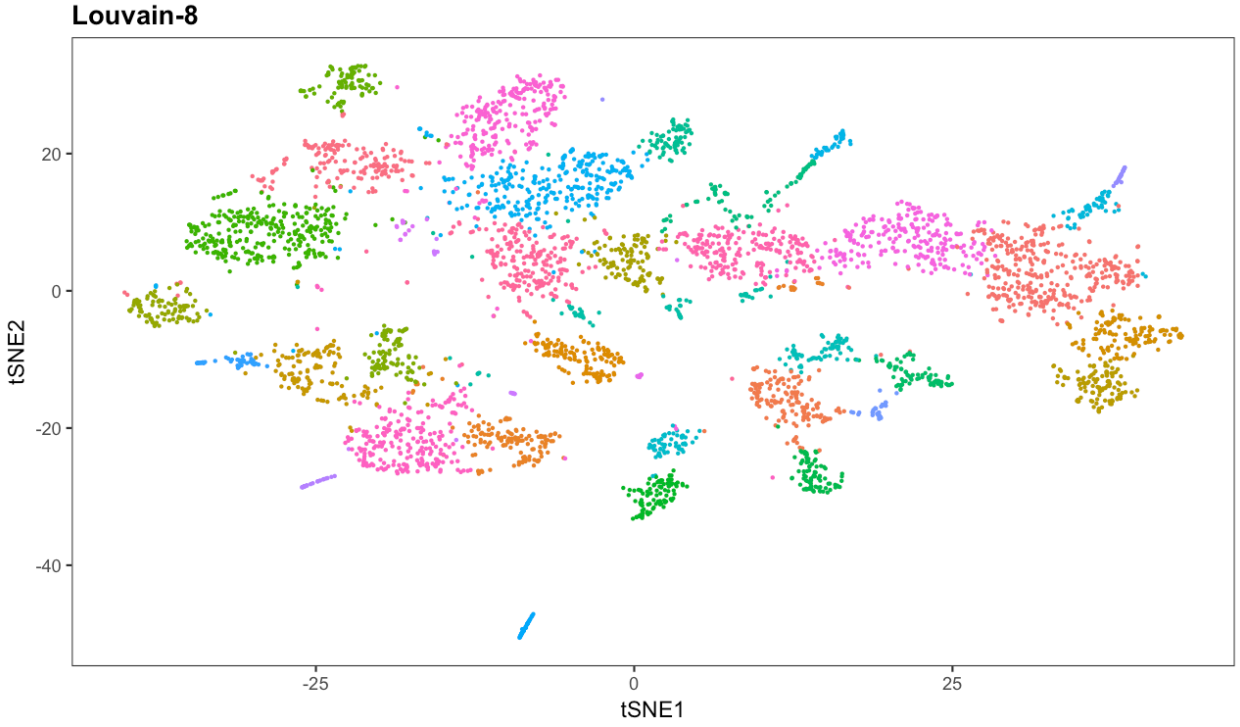
```
# Do graph clustering with Louvain-Jaccard
object.6s.mnn <- graphClustering(object.6s.mnn, num.nn = c(5, 8, 10, 15), method = "Louvain",
    do.jaccard = T)

# Do graph clustering with Infomap-Jaccard
object.6s.mnn <- graphClustering(object.6s.mnn, num.nn = c(10, 15, 20, 30, 40), method = "Infomap",
    do.jaccard = T)
```

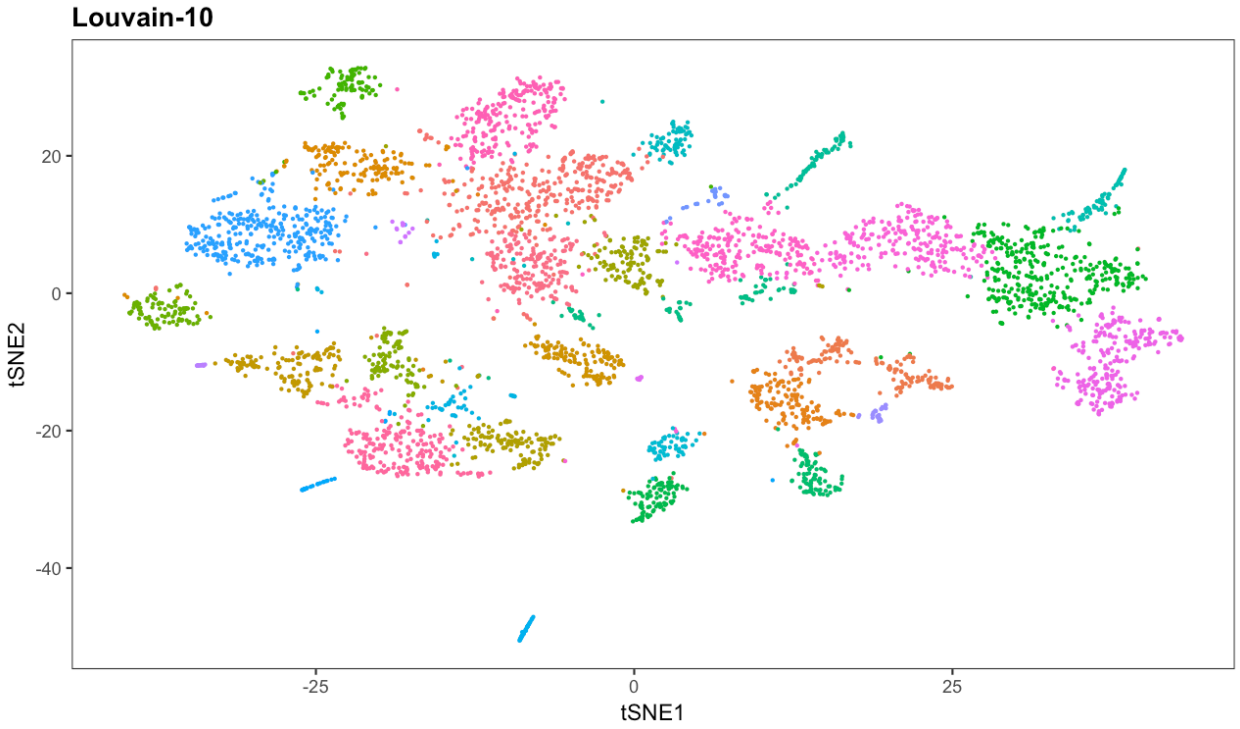
Plot individual clusterings

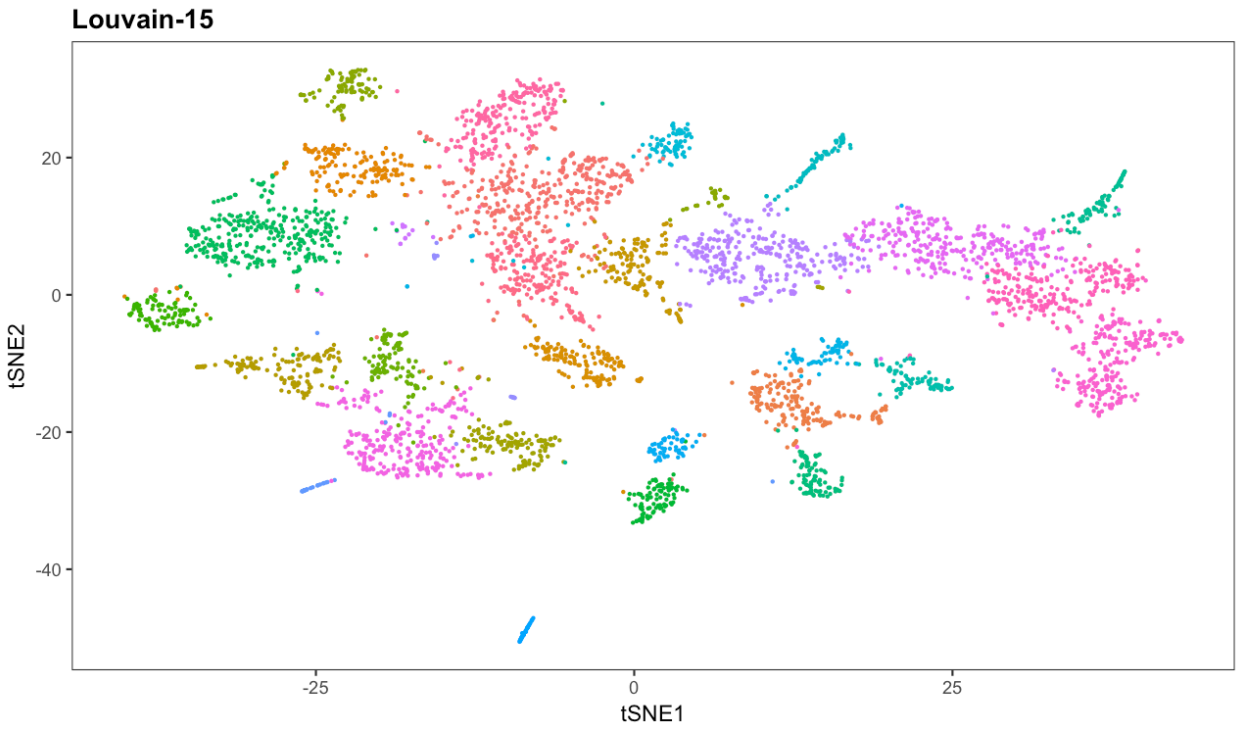
Many of these parameter choices seems fairly valid. Infomap-20 and Infomap-30 clusterings have a resolution that seems reasonable, given our expectations for the number of cell populations present in this stage, and seem to draw boundaries that agree with the most dramatic boundaries in the tSNE plot.

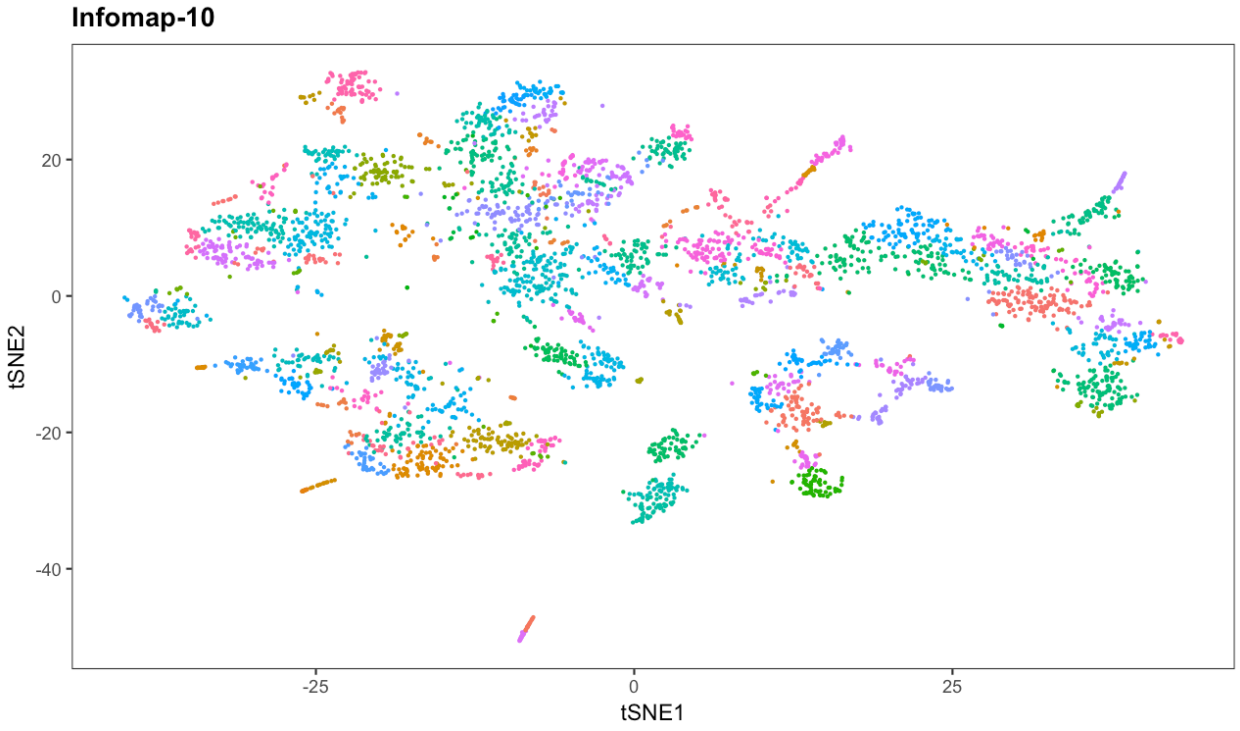
```
clusterings <- c(paste0("Louvain-", c(5, 8, 10, 15)), paste0("Infomap-", c(10, 15,
    20, 30, 40)))
for (c in clusterings) {
    plot(plotDim(object.6s.mnn, c, legend = F))
}
```

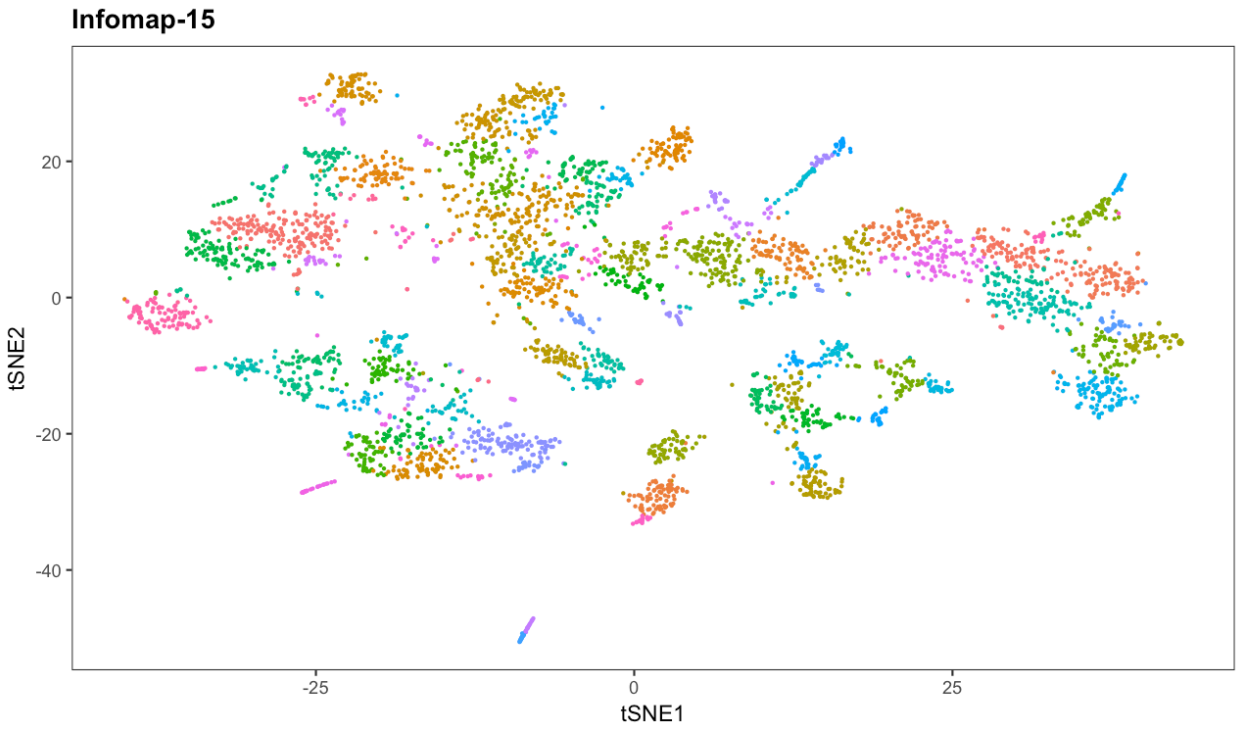


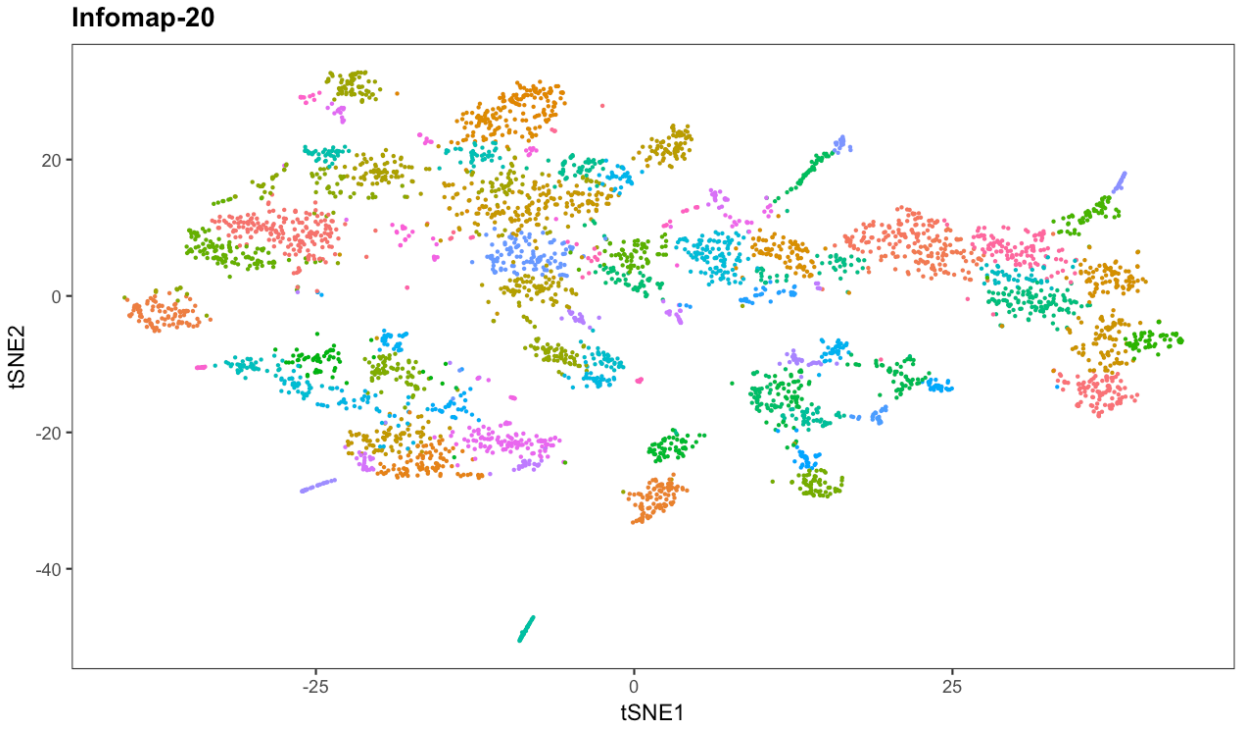


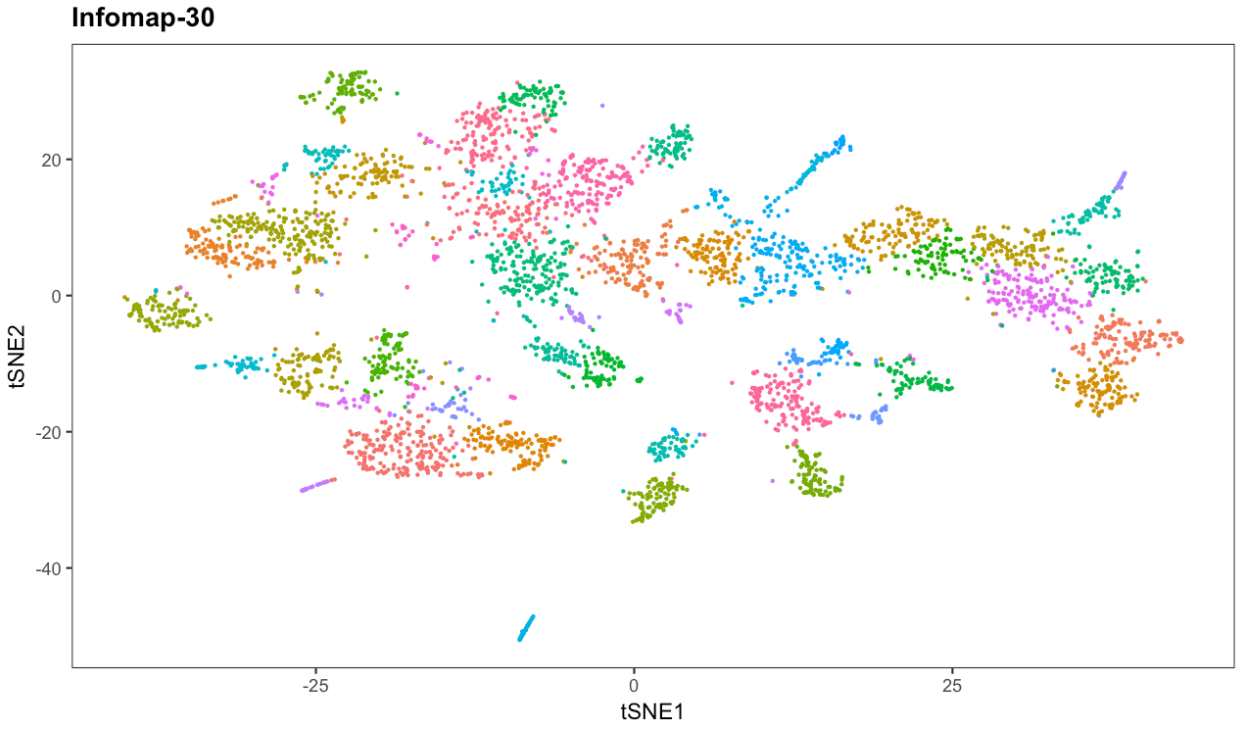

Louvain-10
20
0
-20
-40
tSNE2
-25
0
25
tSNE1

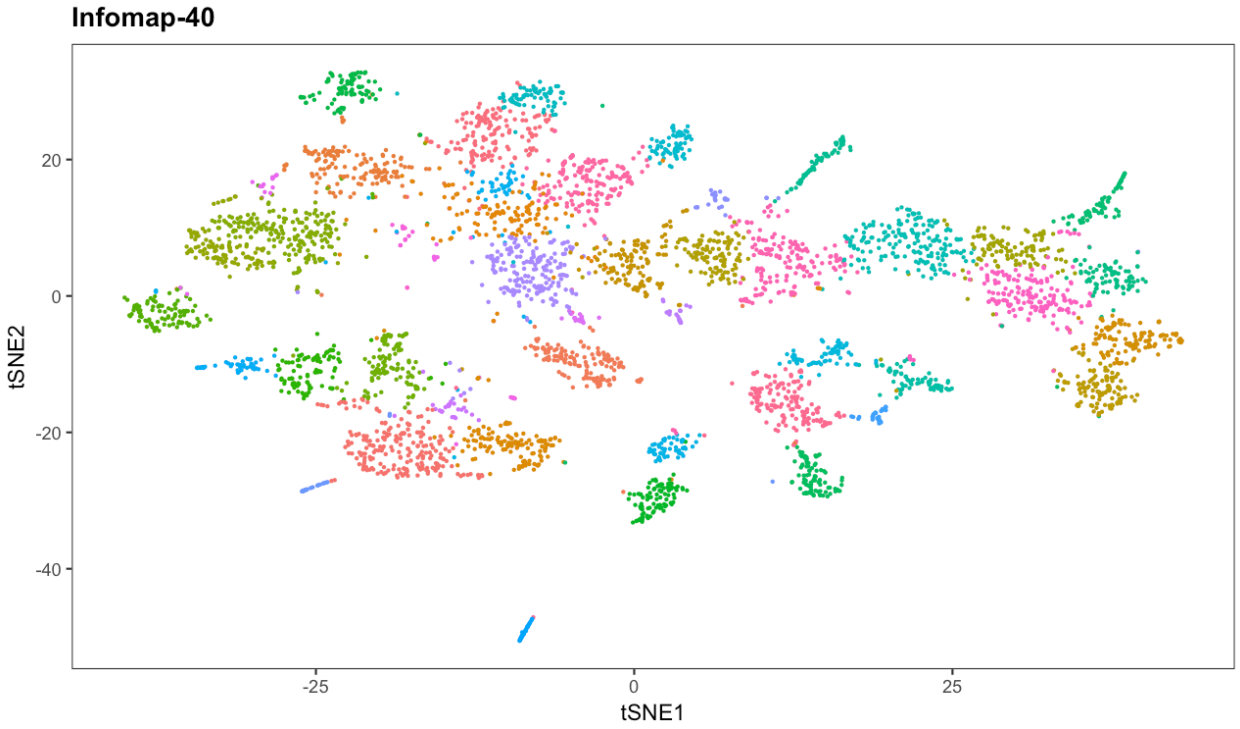

Louvain-15
20
0
-20
-40
tSNE2
-25
0
25
tSNE1


Infomap-10
20
0
-20
-40
tSNE2
-25
0
25
tSNE1


Infomap-15
20
0
-20
-40
tSNE2
-25
0
25
tSNE1


Infomap-20
20
0
-20
-40
tSNE2
-25
0
25
tSNE1


Infomap-30
20
0
-20
-40
tSNE2
-25
0
25
tSNE1



Markers of each cluster

We calculated the top markers of each cluster in the data, and assigned cluster identities based on the known expression patterns of some of these top markers.

```
clusters <- unique(object.6s.mnn@group.ids$`Infomap-30`)

# Precision-recall markers to find the best 'markers' of each cluster.
pr.markers <- lapply(clusters, function(c) markersAUCPR(object.6s.mnn, clust.1 = c,
    clustering = "Infomap-30", genes.use = object.6s.mnn@var.genes))
names(pr.markers) <- clusters
```

Assign clusters

A totally independent clustering could be attempted here, but since there is so much prior knowledge in zebrafish, we use it to evaluate and annotate our clustering.

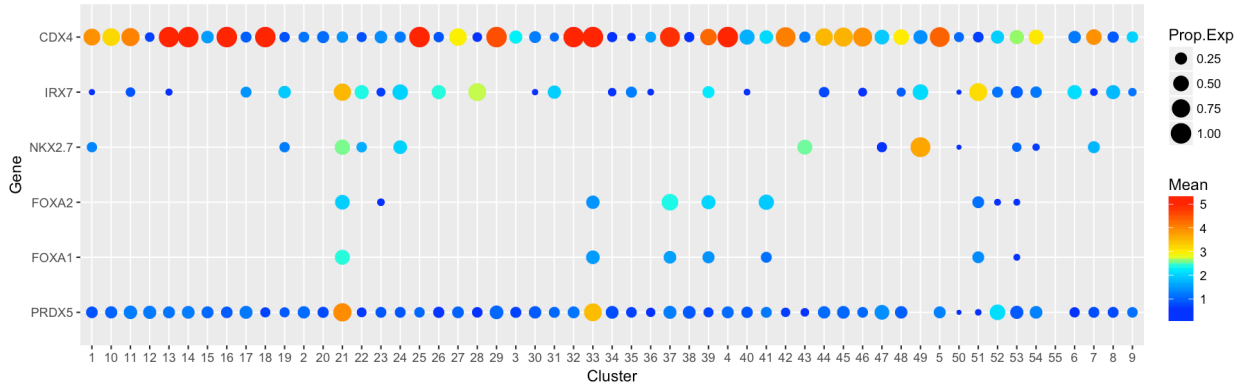
First, we created data.frames to keep track of our cluster assignments.

```
# Make a set of data.frames to keep track during cluster assignment.
I30.n <- length(unique(object.6s.mnn@group.ids$`Infomap-30`))
I30.cluster.assignments <- data.frame(cluster = 1:I30.n, name = rep(NA, I30.n), tip = rep(NA,
    I30.n), row.names = 1:I30.n)

I20.n <- length(unique(object.6s.mnn@group.ids$`Infomap-20`))
I20.cluster.assignments <- data.frame(cluster = 1:I20.n, name = rep(NA, I20.n), tip = rep(NA,
    I20.n), row.names = 1:I20.n)
```

Endoderm

Markers of cluster 21 and 33 include the endoderm markers PRDX5, FOXA1, and FOXA2. 21 expresses pharyngeal endoderm markers NKX2.7 and IRX7, while 33 expresses the posterior marker CDX4, marking it as the pancreatic and interstinal endoderm.

```
plotDot(object.6s.mnn, genes = c("PRDX5", "FOXA1", "FOXA2", "NKX2.7", "IRX7", "CDX4"),
    clustering = "Infomap-30")
```

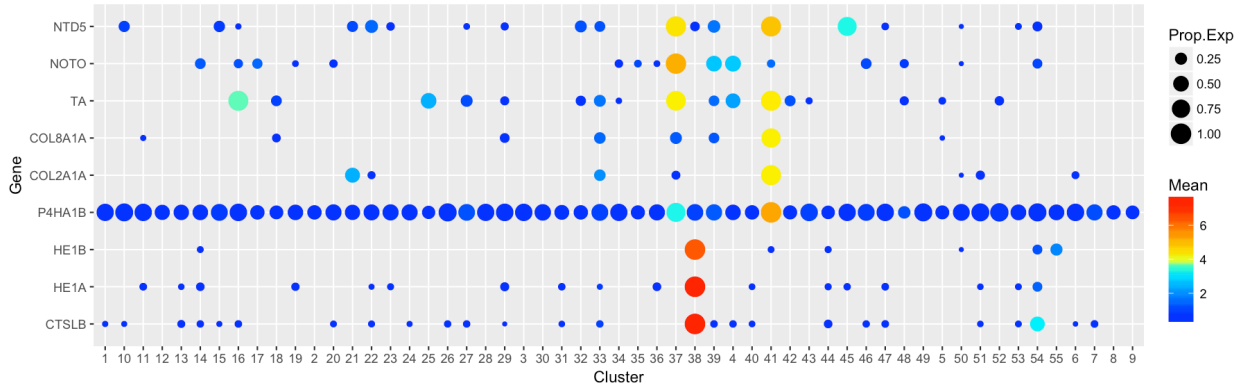



```
I30.cluster.assignments["21", "name"] <- "Endoderm Pharyngeal"
I30.cluster.assignments["33", "name"] <- "Endoderm Pancreatic/Intestinal"
```

Axial mesoderm

Cluster 38 is the prechordal plate, based on its expression of hatching enzymes (HE1A, HE1B, and CTSLB/HGG1). Clusters 37 and 41 are the notochord, based on their expression of the collagens (COL2A1A, COL8A1A), collagen synthesis related enzymes (P4HA1B), and classic notochord genes (TA/NTL, NOTO/FLH, and NTD5).

```
plotDot(object.6s.mnn, genes = c("CTSLB", "HE1A", "HE1B", "P4HA1B", "COL2A1A", "COL8A1A",
    "TA", "NOTO", "NTD5"), clustering = "Infomap-30")
```

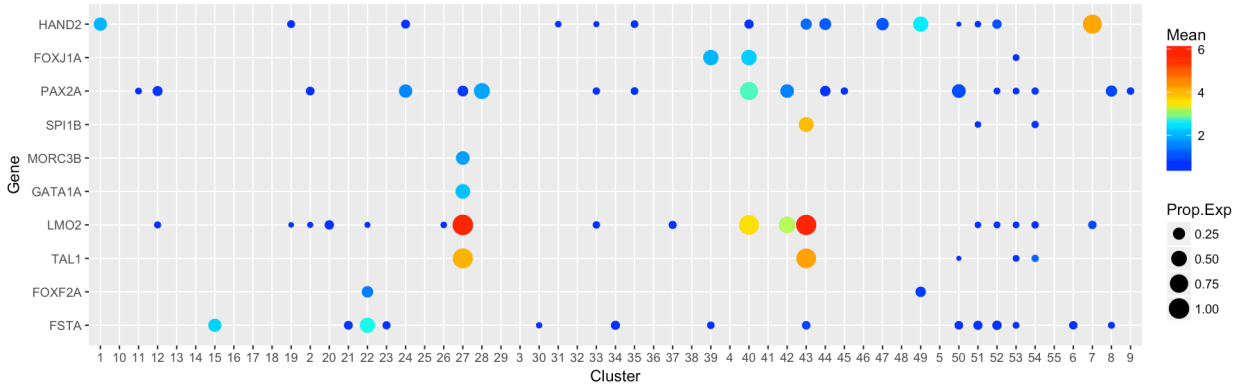



```
# 38 is Prechordal plate due to expression of hatching enzymes HE1B, HE1A, CTSLB
# (hgg1)
I30.cluster.assignments["38", "name"] <- "Prechordal Plate"

# 38 and 41 are the notochord (NOTO, NTD5) 41 seems like the real tip, because
# its strongest markers are the differentiation genes (P4HA1B, COL8A1A, COL9A3,
# PLOD1A, COL2A1A) -- all involved in collagen synthesis
I30.cluster.assignments["37", "name"] <- "Notochord Posterior"  # Don't use as tip
I30.cluster.assignments["41", "name"] <- "Notochord Anterior"  # Use as a tip
```

Intermediate/lateral mesoderm

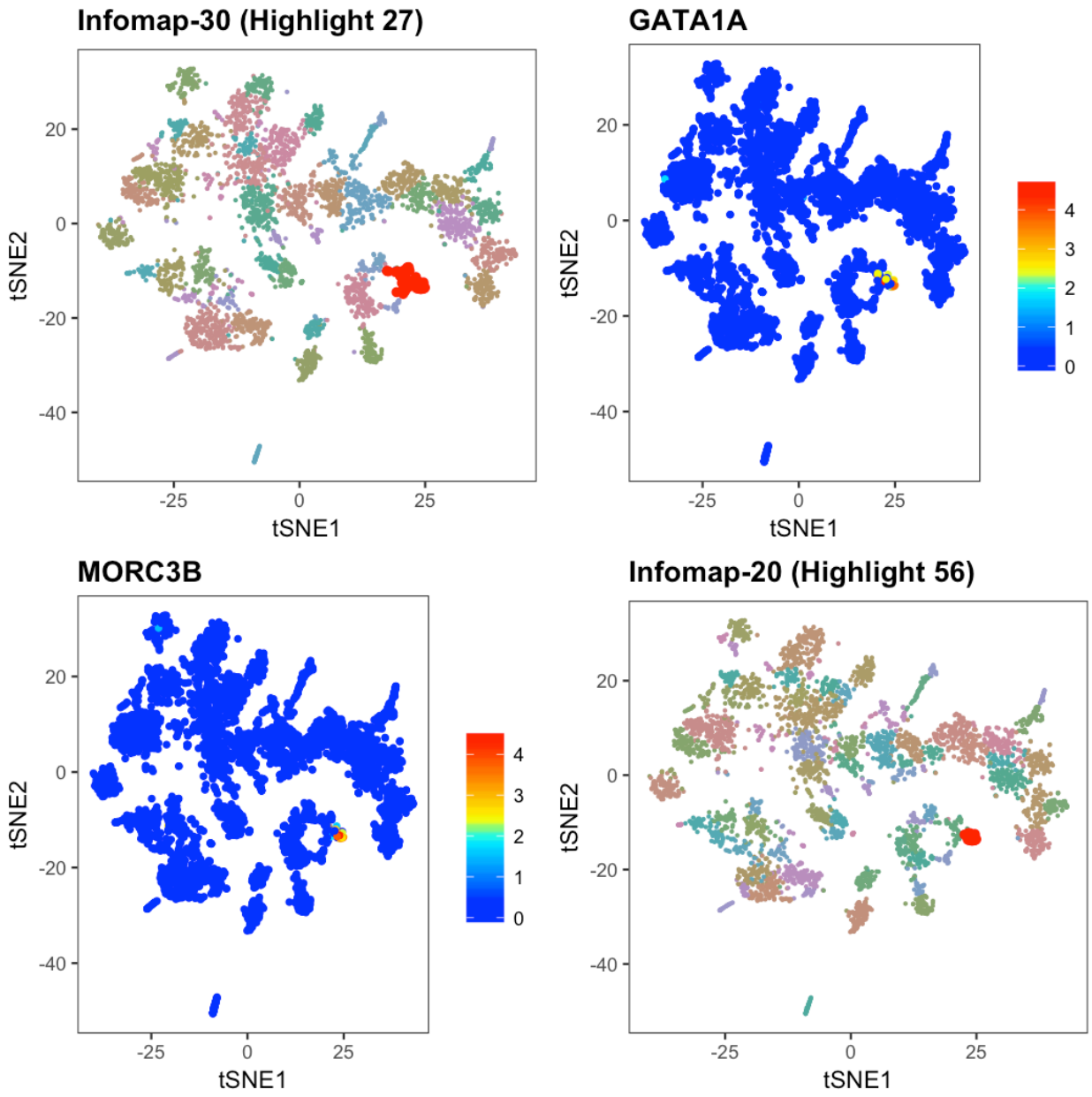
Cluster 22 is the cephalic mesoderm, based on its expression of the markers FOXF2A and FSTA. Clusters 27 and 43 are both hematopoietic lineages, based on their expression of TAL1 and LMO2. Cluster 27 is likely the Intermediate Cell Mass (erythroid lineage) based on its expression of GATA1A, whereas cluster 43 is likely the Rostral Blood Island (myeloid lineage) based on its expression of SPI1B, an early macrophage marker. Cluster 40 is the pronephric progenitors, based on its expresion of FOXJ1A and PAX2A. Finally, cluster 7 is the heart primordium, based on its expression of the classic marker HAND2.

```
plotDot(object.6s.mnn, genes = c("FSTA", "FOXF2A", "TAL1", "LMO2", "GATA1A", "MORC3B",
    "SPI1B", "PAX2A", "FOXJ1A", "HAND2"), clustering = "Infomap-30")
```



However, the boundary of cluster 27 doesn't agree with the boundary of expression of the classic markers of this cell type (GATA1A and MORC3B, for instance). Cluster 56 from the finer resolution Infomap-20 clustering agrees with gene expression more, so we will use that for Hematopoietic (ICM).

```
grid.arrange(grobs = list(plotDimHighlight(object.6s.mnn, "Infomap-30", "27", legend = F),
    plotDim(object.6s.mnn, "GATA1A"), plotDim(object.6s.mnn, "MORC3B"), plotDimHighlight(object.6s.mnn,
        "Infomap-20", "56", legend = F)))
```

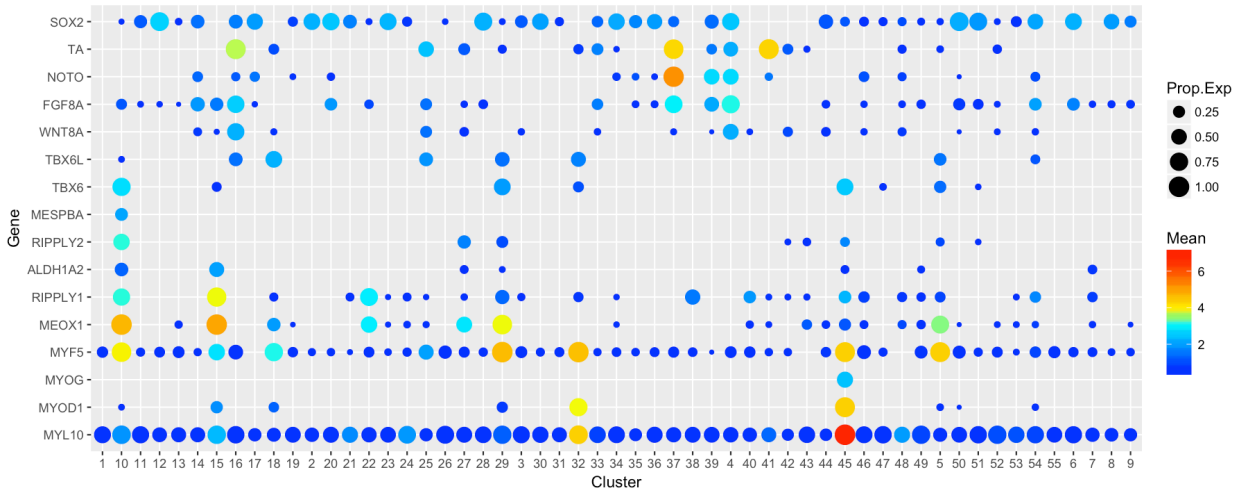



```
# 22 is cephalic mesoderm?? FSTA, FOXF2A
I30.cluster.assignments["22", "name"] <- "Cephalic Mesoderm"

# 43 is Hematopoietic (TAL1, LMO2), Rostral Blood Island (SPI1B)
I30.cluster.assignments["43", "name"] <- "Hematopoeitic (RBI)"

# Infomap-20 Cluster 56 seems to be a better GATA1A+ cluster.
I20.cluster.assignments["56", "name"] <- "Hematopoeitic (ICM)"

# 40 seems to be pronephros (PAX2A + FOXJ1A)
I30.cluster.assignments["40", "name"] <- "Pronephros"

# 7 is clearly the heart primordium (expression of HAND2, GATA6, and GATA5)
I30.cluster.assignments["7", "name"] <- "Heart Primordium"
```

Paraxial mesoderm

Clusters 32 and 45 are the adaxial cells, based on their expression of MYL10, MYOD1, and MYOG. (Based on MYOG, cluster 45 is the more differentiated group of adaxial cells.) Clusters 10 and 15 are the somites, based on expression of MEOX1, RIPPLY1 and ALDH1A2. Cluster 10 is likely the forming simutes, based on its continued expression of MESPBA, RIPPLY2 (which is expressed in S-II and S-I), and continued expression of TBX6. Clusters 25, 18, 5, and 29 are the pre-somitic mesoderm, based on their expression of TBX6L. Finally, clusters 4 and 16 are tailbud, based on their overlapping expression of SOX2, TA, NOTO, FGF8A, and WNT8A.

```
plotDot(object.6s.mnn, genes = c("MYL10", "MYOD1", "MYOG", "MYF5", "MEOX1", "RIPPLY1",
    "ALDH1A2", "RIPPLY2", "MESPBA", "TBX6", "TBX6L", "WNT8A", "FGF8A", "NOTO", "TA",
    "SOX2"), clustering = "Infomap-30")
```

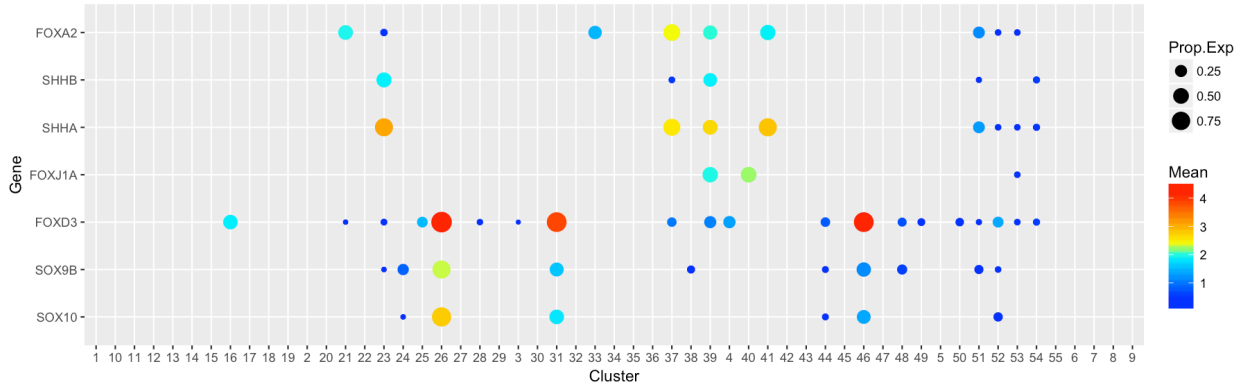



```
# 45 and 32 are the adaxial cells; 45 seems like the real tip.  (ACTA1A, MYL10,
# ACTC1B, ACTC1A, MEF2D, MYOG)
I30.cluster.assignments["32", "name"] <- "Adaxial Cells"  # Don't use as tip
I30.cluster.assignments["45", "name"] <- "Adaxial Cells"  # Use as the tip

# 10 and 15 is the formed somites (MEOX1, RIPPLY1, ALDH1A2)
markers.10v15 <- markersAUCPR(object.6s.mnn, clust.1 = "10", clust.2 = "15", clustering = "Infomap-30")
# Ripply1 mostly in formed somites, Ripply2 mostly in S-I and S-II MESPBA in
# future anterior half of S-II and S-I
I30.cluster.assignments["10", "name"] <- "Somites Forming"  # Don't use as tip
I30.cluster.assignments["15", "name"] <- "Somites Formed"  # Use as a tip

# 25/18/5/29 are the PSM.
I30.cluster.assignments["25", "name"] <- "PSM Maturation Zone"  # Not a tip
I30.cluster.assignments["18", "name"] <- "PSM Posterior"  # Not a tip
I30.cluster.assignments["5", "name"] <- "PSM Intermediate"  # Not a tip
I30.cluster.assignments["29", "name"] <- "PSM Intermediate"  # Not a tip

# Clusters 4 and 16 seems to be the tailbud, based on its expression of WNT8A,
# FGF8A, NOTO, TA, SOX2. They represent, to some degree the more neural inclined
# and more mesoderm inclined tissues (i.e. I think some of the early
# differentiation is also in there.)
I30.cluster.assignments["4", "name"] <- "Tailbud"  # Use as a tip
I30.cluster.assignments["16", "name"] <- "Tailbud"  # Use as a tip
```

Neural

Clusters 26, 41, and 31 are the neural crest, based on their expression of FOXD3, SOX9B, and SOX10. Cluster 39 seems to be the floor plate, based on its combined expression of SHHA, SHHB, FOXJ1A, and FOXA2.

```
plotDot(object.6s.mnn, genes = c("SOX10", "SOX9B", "FOXD3", "FOXJ1A", "SHHA", "SHHB",
    "FOXA2"), clustering = "Infomap-30")
```

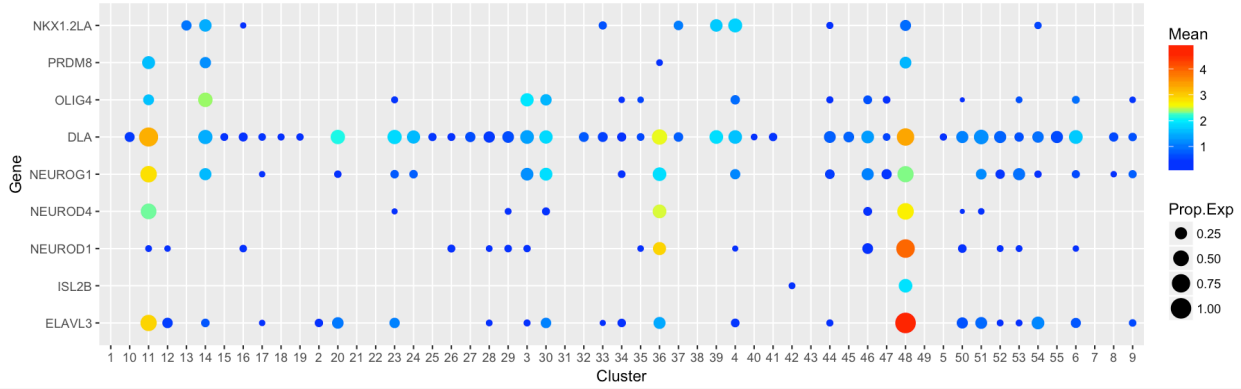



```
# 26, 46, 31 are neural crest lineages given their high expression of SOX10 and
# FOXD3.
I30.cluster.assignments["26", "name"] <- "Neural Crest"  # Tip
I30.cluster.assignments["46", "name"] <- "Neural Crest Forming"  # Not tip
I30.cluster.assignments["31", "name"] <- "Neural Crest Forming"  # Not tip

# Cluster 39 -- floor plate???  (FOXJ1A, SHHA, SHHB, FOXA2)
I30.cluster.assignments["39", "name"] <- "Floor Plate"  # Try as a tip?
```

Spinal Cord

Clusters 48, 11, and 14 are spinal cord. 48 and 11 are the more differenciated, given their high expression of ELAVL3, NEUROD1, NEUROD4, and NEUROG4.

```
plotDot(object.6s.mnn, genes = c("ELAVL3", "ISL2B", "NEUROD1", "NEUROD4", "NEUROG1",
    "DLA", "OLIG4", "PRDM8", "NKX1.2LA"), clustering = "Infomap-30")
```

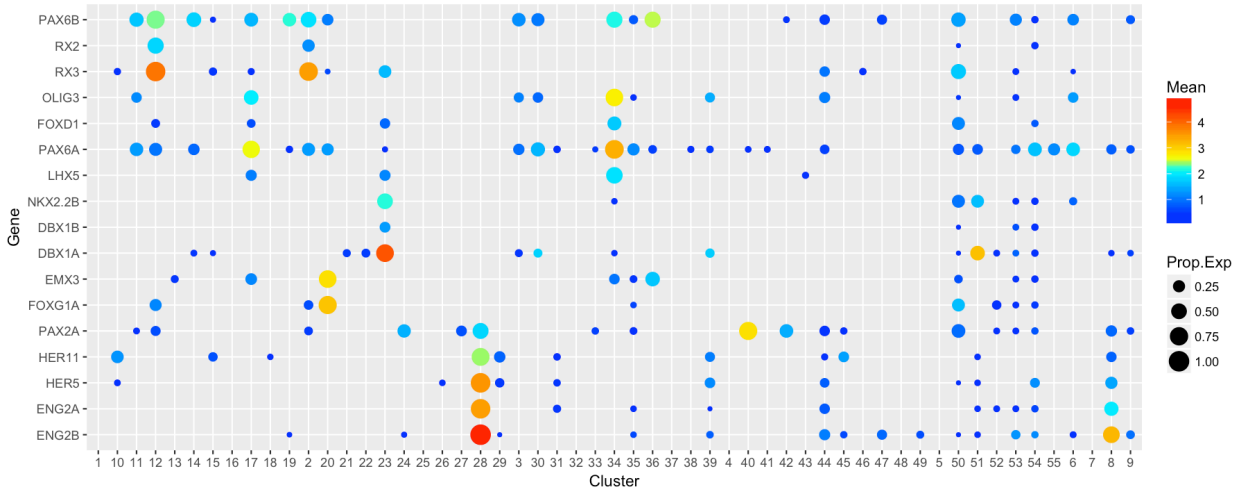



```
# Cluster 48 -- Spinal Cord ELAVL3, ISL2B, NEUROD1
I30.cluster.assignments["48", "name"] <- "Spinal Cord Differentiated"  # Tip?

# Cluster 11 -- Also spinal cord NEUROD4, NEUROG1, ELAVL3, DLA Difference vs. 48
# = higher SOX3, SOX19A (just more progenitor like)
I30.cluster.assignments["11", "name"] <- "Spinal Cord"  # Tip?

# Cluster 14 -- Spinal Cord Progenitors CHD, HER3, OLIG4, PRDM8, NKX1.2LA
I30.cluster.assignments["14", "name"] <- "Spinal Cord Progenitors"  # Not tip
```

Fore/mid-brain

Clusters 28 and 8 are the midbrain, given their expression of ENG2A, ENG2B, HER5, HER11, and PAX2A. Cluster 20 is the telencephalon, given its expression of FOXG1A and EMX3. Clusters 12 and 2 are the optic cup, given their expression of RX3, RX2, and PAX6B. Cluster 23 is the ventral diencephalon, given its expression of DBX1A, DBX1B, and NKX2.2B. Finally, clusters 17 and 34 are the dorsal diencephalon, given their expression of LHX5, PAX6A, FOXD1, and OLIG3.

```
plotDot(object.6s.mnn, genes = c("ENG2B", "ENG2A", "HER5", "HER11", "PAX2A", "FOXG1A",
    "EMX3", "DBX1A", "DBX1B", "NKX2.2B", "LHX5", "PAX6A", "FOXD1", "OLIG3", "RX3",
    "RX2", "PAX6B"), clustering = "Infomap-30")
```



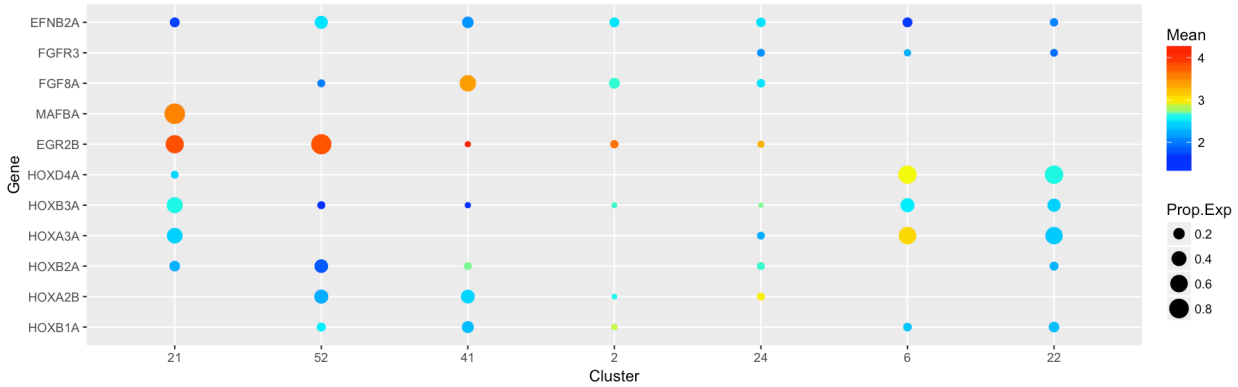

```
# Cluster 28 & 8 Midbrain ENG2B, ENG2A, HER5, HER11, PAX2A
I30.cluster.assignments["28", "name"] <- "Midbrain"  # Use as tip
I30.cluster.assignments["8", "name"] <- "Midbrain"  # Don't use as tip

# 20 - Telencephalon FOXG1A / EMX3
I30.cluster.assignments["20", "name"] <- "Telencephalon"  # Use as a tip

# 23 - Ventral Diencephalon NKX2.4B / NKX2.4A / NKX2.1 / DBX1A / DBX1B / SHHA /
# NKX2.2B
I30.cluster.assignments["23", "name"] <- "Diencephalon Ventral"  # Don't use as a tip

# 34 / 17 - Dorsal Diencephalon ARX / LHX5 / PAX6A / FOXD1 / OLIG3 / OLIG2 Major
# differences between the clusters seem to be cell cycle, translation related, so
# think these clusters should be combined.
I30.cluster.assignments["34", "name"] <- "Dorsal Diencephalon"  # Use as a tip
I30.cluster.assignments["17", "name"] <- "Dorsal Diencephalon"  # Use as a tip

# 12 / 2 - Optic Vesicle RX3 / RX2 / HMX4 / PAX6B / MAB21L2 Differential markers
# are cell cycle related; think these clusters should be combined.
I30.cluster.assignments["12", "name"] <- "Optic Cup"  # Use as a tip, but combined
I30.cluster.assignments["2", "name"] <- "Optic Cup"  # Use as a tip
```

Hindbrain

It seems that for the hindbrain, Infomap-30 doesn't have enough resolution, based on the cluster boundaries and the boundaries of expression of the best cluster markers. Going to use Infomap-20 for clustering in this region.

It seems that cluster 21 is rhombomeres 5 & 6 (given its expression of MAFBA/VAL). Thus cluster 52 must be rhombomere 3 (given its expression of EGR2B/KROX20 and that cluster 21 contains rhombomere 5). Cluster 41 is rhombomere 4, since it expresses FGF8A. Clusters 6 and 22 are rhombomere 7, given their expression of HOXD4A and HOXA3A.

```
i20.hb.clusters <- c("21", "52", "41", "2", "24", "6", "22")

# Try Infomap-20 clusters for hindbrain?
plotDot(object.6s.mnn, genes = c("HOXB1A", "HOXA2B", "HOXB2A", "HOXA3A", "HOXB3A",
    "HOXB4A", "HOXD4A", "EGR2B", "MAFBA", "FGF8A", "FGFR3", "EFNB2A"), clustering = "Infomap-20",
    mean.expressing.only = T, clusters.use = i20.hb.clusters)
```

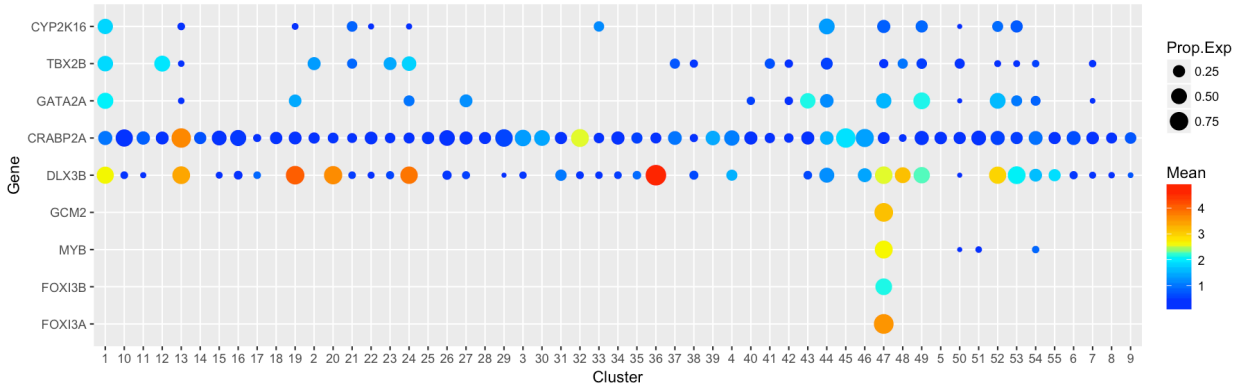



```
# 21 is rhombomere 5/6
I20.cluster.assignments["21", "name"] <- "Hindbrain R5+6"

# 52 is rhombomere 3
I20.cluster.assignments["52", "name"] <- "Hindbrain R3"

# 41 is rhombomere 4
I20.cluster.assignments["41", "name"] <- "Hindbrain R4"

I20.cluster.assignments["2", "name"] <- "Hindbrain"  # Don't use as tip
# (What exactly is this? Some kind of as yet unpatterned hindbrain progenitors?)

# 6 and 22 are rhombomere 7?
I20.cluster.assignments["6", "name"] <- "Hindbrain R7"
I20.cluster.assignments["22", "name"] <- "Hindbrain R7"
```

Non-neural ectoderm

Cluster 47 is the integument, given its expression of FOXI3A, FOXI3B, MYB, and GCM2. Cluster 13 is the neural plate border, given its expression of CRABP2A and DLX3B. Cluster 1 is the epidermis given its expression of GATA2A, TBX2B, and CYP2K16.

```
plotDot(object.6s.mnn, genes = c("FOXI3A", "FOXI3B", "MYB", "GCM2", "DLX3B", "CRABP2A",
    "GATA2A", "TBX2B", "CYP2K16"), clustering = "Infomap-30")
```



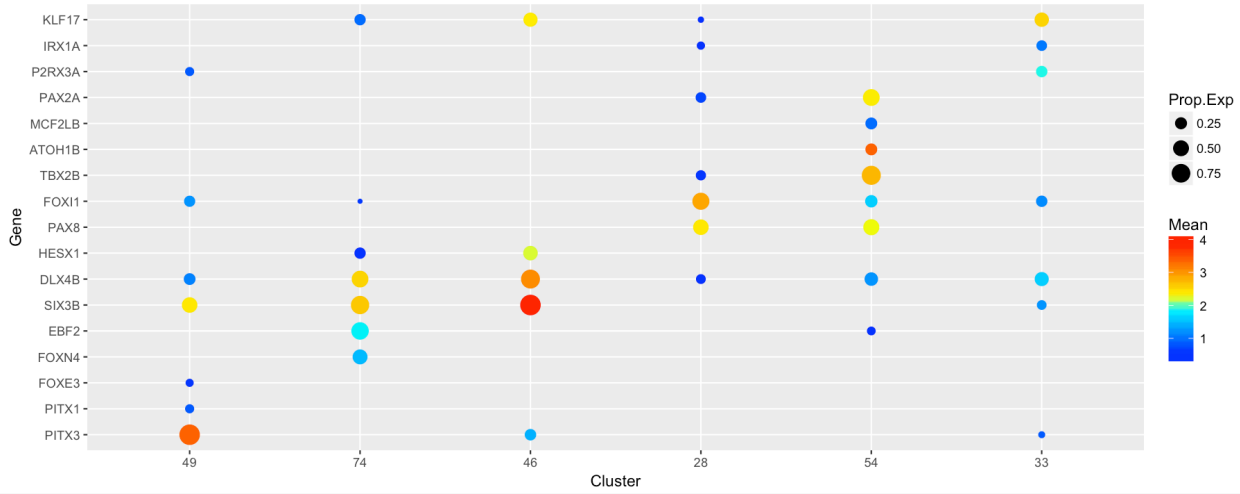

```
# 47 is integument?  FOXI3A / FOXI3B / MYB / GCM2
I30.cluster.assignments["47", "name"] <- "Integument"  # Use as a tip

# Non-neural ectoderm Cluster 13 is the neural plate border
I30.cluster.assignments["13", "name"] <- "Neural Plate Border"  # Use as a tip

# Cluster 1 is the epidermis
I30.cluster.assignments["1", "name"] <- "Epidermis"  # Use as a tip
```

Pre-placodal ectoderm

Down in the pre-placodal ectoderm / placode territory, it seems like Infomap-30 doesn't really have enough resolution, based on the cluster boundaries vs. the expression domains of the top markers of those clusters. Thus, going to use Infomap-20 clusters for this region.

Cluster 49 is the lens placode (PITX3+). Cluster 74 is olfactory placode (FOXN4+, EBF2/COE+, SIX3B+, DLX4B+). Cluster 46 is the adenohypophyseal placode (stronger SIX3B, DLX4B, HESX1). Cluster 28 is the epibranchail placode (FOXI1+, PAX8+). Cluster 54 is the otic placode (ATOH1B, TBX2B, PAX2A). Cluster 33 is the trigeminal placode (KLF17, IRX1A, P2RX3A).

```
plotDot(object.6s.mnn, genes = c("PITX3", "PITX1", "FOXE3", "FOXN4", "EBF2", "SIX3B",
    "DLX4B", "HESX1", "PAX8", "FOXI1", "TBX2B", "ATOH1B", "MCF2LB", "PAX2A", "P2RX3A",
    "IRX1A", "KLF17"), clustering = "Infomap-20", clusters.use = c("49", "74", "46",
    "28", "54", "33"))
```

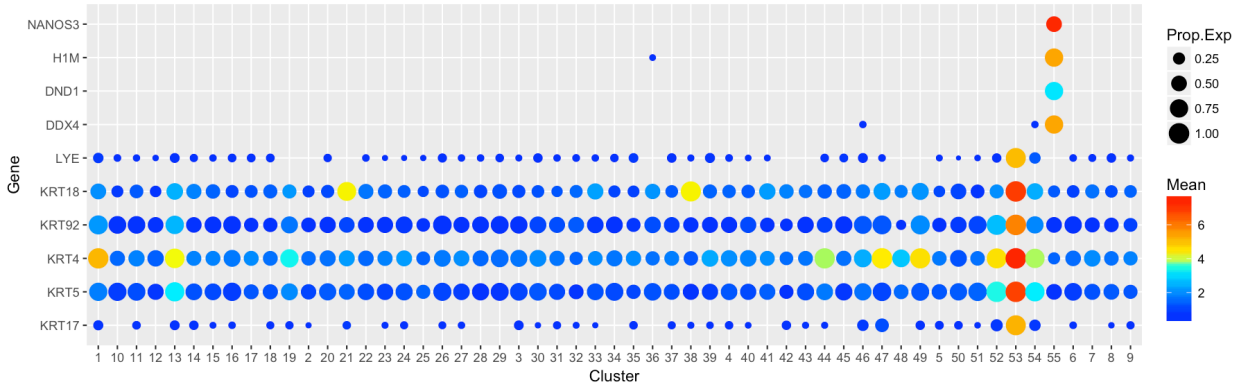



```
# Looks like 49 really is the lens.  (PITX3, PITX1, SIX7, FOXE3)
I20.cluster.assignments["49", "name"] <- "Placode Lens"

# 74 is olfactory (FOXN4, EBF2, PRDM8, GATAD2B)
I20.cluster.assignments["74", "name"] <- "Placode Olfactory"

# 46 is adenohypophyseal SIX3B, DLX4B, DLX3B, HESX1,
I20.cluster.assignments["46", "name"] <- "Placode Adenohypophyseal"

# 28 is/includes epibranchial -- seems right from PAX8 FOXI1, PAX8, FOXI1,
# PRDM12B, NKX2.3, GBX2
I20.cluster.assignments["28", "name"] <- "Placode Epibranchial"

# 54 is otic ATOH1B? MCF2LB? STC2A? GBX2? ROBO4? DLX3B? SOX9A?  (otic) (otic)
# TBX2B ATOH1B PAX2A SOX9A
I20.cluster.assignments["54", "name"] <- "Placode Otic"

# 33 is trigeminal placode P2RX3A IRX1A KLF17
I20.cluster.assignments["33", "name"] <- "Placode Trigeminal"
```

Non-blastoderm

Cluster 53 is the EVL / Periderm (all of the keratins!). Cluster 55 is the PGCs (DDX4, H1M, NANOS3). Cluster 54 seems to be mostly YSL-related markers, and should not be used in tree-building.

```
plotDot(object.6s.mnn, genes = c("KRT17", "KRT5", "KRT4", "KRT92", "KRT18", "LYE",
    "DDX4", "DND1", "H1M", "NANOS3"), clustering = "Infomap-30")
```



The EVL population seems to need to be cleaned up however – some cells are really good expressers of the EVL markers, whereas others are not.

```
evl.score <- apply(object@logupx.data[c("LYE", "KRT18", "KRT92", "KRT4", "KRT5",
    "KRT17"), cellsInCluster(object.6s.mnn, "Infomap-30", "53")], 2, sum.of.logs)

new.evl <- names(which(evl.score > 9))
remove.evl <- names(which(evl.score <= 9))
# 53 - EVL/Periderm KRT17, KRT5, KRT4, KRT92, KRT18, LYE FOXI3A / FOXI3B / MYB /
# GCM2
I30.cluster.assignments["53", "name"] <- "EVL/Periderm"  # Use as a tip

# 55 - PGCs DDX4 / H1M / NANOS3
I30.cluster.assignments["55", "name"] <- "Primordial Germ Cells"

# 54 - YSL/yolk APOA1B / PVALB9 / SEPP1A / CTSLL Top markers are not particularly
# great markers and are either ubiquitous or yolk-associated. This is
# contamination.
I30.cluster.assignments["54", "name"] <- "YSL"  # Don't use as a tip
```

Generate final clusterings

```
# Combine clustering assignments from two clusterings
I30.cluster.assignments$clustering <- "Infomap-30"
I20.cluster.assignments$clustering <- "Infomap-20"
cluster.assignments <- rbind(I30.cluster.assignments, I20.cluster.assignments)

# Remove any clusters that weren't assigned an identity
cluster.assignments <- cluster.assignments[!is.na(cluster.assignments$name), ]

# Renumber clusters
cluster.assignments$cluster.new <- 1:nrow(cluster.assignments)

# Create blank clusterings in the 6-somite object
object.6s.mnn@group.ids$clusters.6s.name <- NA
object.6s.mnn@group.ids$clusters.6s.num <- NA

# Copy cell identities over for each cluster
for (i in 1:nrow(cluster.assignments)) {
    cells <- cellsInCluster(object.6s.mnn, clustering = cluster.assignments[i, "clustering"],
        cluster = cluster.assignments[i, "cluster"])
    object.6s.mnn@group.ids[cells, "clusters.6s.name"] <- cluster.assignments[i,
        "name"]
    object.6s.mnn@group.ids[cells, "clusters.6s.num"] <- as.character(cluster.assignments[i,
        "cluster.new"])
}

# Remove the bad cells from cluster 53 that aren't EVL.
object.6s.mnn@group.ids[remove.evl, "clusters.6s.name"] <- NA
object.6s.mnn@group.ids[remove.evl, "clusters.6s.num"] <- NA
```

Transfer clusterings to main object

Need to transfer cluster identities from the 6-somite only object to the full object.

```
object@group.ids$`ZF6S-Infomap-30` <- NA
object@group.ids[rownames(object.6s.mnn@group.ids), "ZF6S-Infomap-30"] <- object.6s.mnn@group.ids$`Infomap-30`

object@group.ids$`ZF6S-Infomap-20` <- NA
object@group.ids[rownames(object.6s.mnn@group.ids), "ZF6S-Infomap-20"] <- object.6s.mnn@group.ids$`Infomap-20`

object@group.ids$`ZF6S-Cluster` <- NA
object@group.ids[rownames(object.6s.mnn@group.ids), "ZF6S-Cluster"] <- object.6s.mnn@group.ids$clusters.6s.name

object@group.ids$`ZF6S-Cluster-Num` <- NA
object@group.ids[rownames(object.6s.mnn@group.ids), "ZF6S-Cluster-Num"] <- object.6s.mnn@group.ids$clusters.6s.num
```

Save objects

We save here the 6-somite object, the full object with our 6-somite clustering added to it, and also a data.frame of the tips that can be used during further inspection to annotate which tips should be used in the tree building.

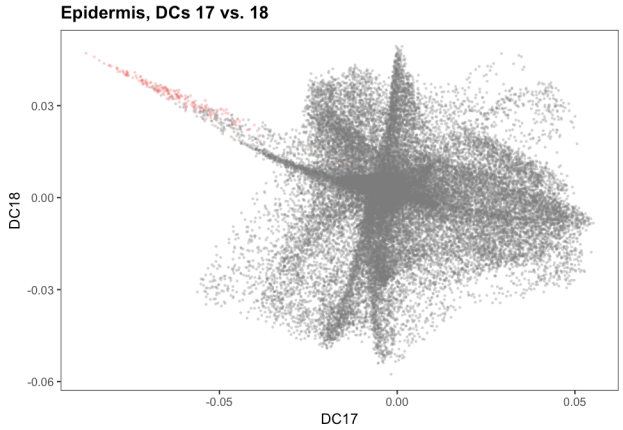
```
saveRDS(object, file = "obj/object_4_withTips.rds")
saveRDS(object.6s.mnn, file = "obj/object_6s.rds")
write.csv(cluster.assignments, file = "dm-plots/tips-use.csv")
```

Plot tips in diffusion map

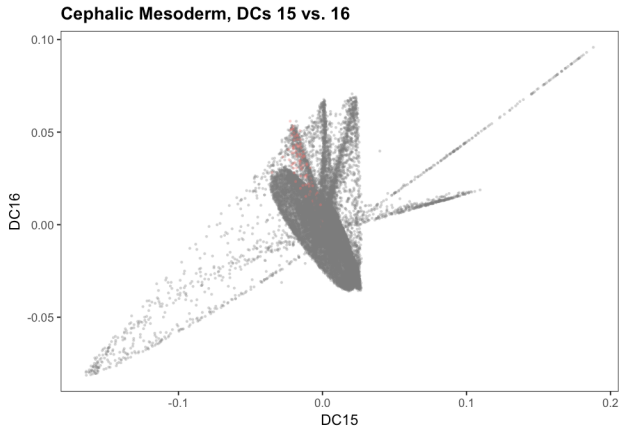
Not all clusters from the final stage should really comprise tips of the developmental tree – progenitor populations that remain should be excluded. For instance, embryos at 6-somite stage contain both somites (a terminal population) and pre-somitic mesoderm that will give rise to additional somites later; the pre-somitic mesoderm should not be a separate tip in the tree.

Here, we show a couple of good plots. For 'bad' clusters that shouldn't be used as tips, all combinations of diffusion components will not separate the cells significantly.

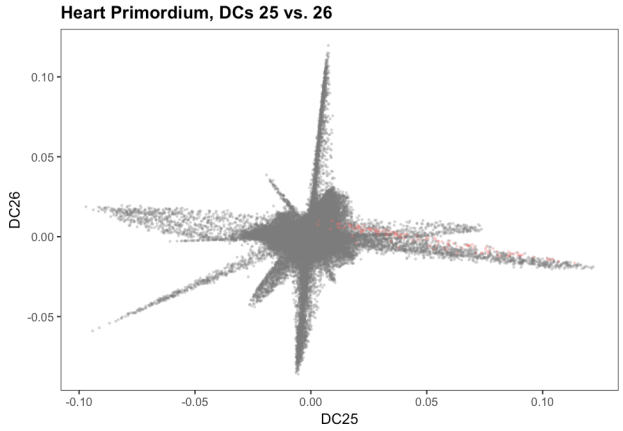
```
object@group.ids$pop <- NA
object@group.ids[cellsInCluster(object, "ZF6S-Cluster", "Epidermis"), "pop"] <- "1"
plotDim(object, label = "pop", plot.title = "Epidermis, DCs 17 vs. 18", reduction.use = "dm",
    dim.x = 17, dim.y = 18, legend = F, alpha = 0.35)
```




```
object@group.ids$pop <- NA
object@group.ids[cellsInCluster(object, "ZF6S-Cluster", "Cephalic Mesoderm"), "pop"] <- "1"
plotDim(object, label = "pop", plot.title = "Cephalic Mesoderm, DCs 15 vs. 16", reduction.use = "dm",
    dim.x = 15, dim.y = 16, legend = F, alpha = 0.35)
```




```
object@group.ids$pop <- NA
object@group.ids[cellsInCluster(object, "ZF6S-Cluster", "Heart Primordium"), "pop"] <- "1"
plotDim(object, label = "pop", plot.title = "Heart Primordium, DCs 25 vs. 26", reduction.use = "dm",
    dim.x = 25, dim.y = 26, legend = F, alpha = 0.35)
```



URD 4: Biased Random Walks

```
library(URD)
```

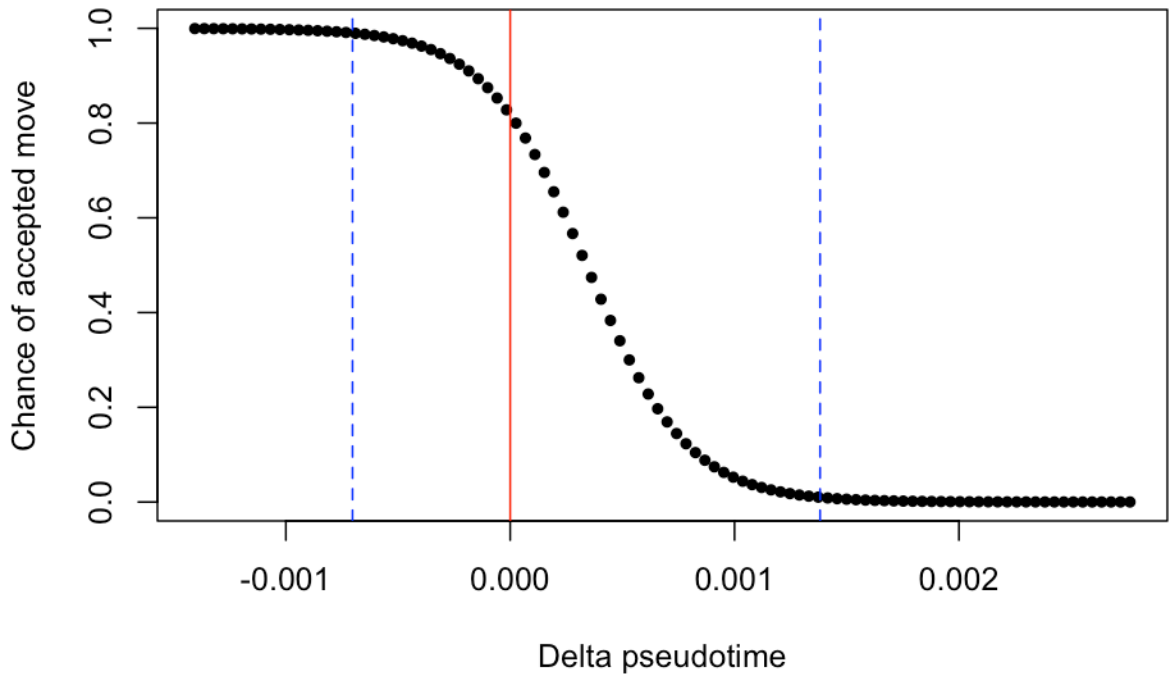
Load previous saved object

```
object <- readRDS("obj/object_4_withTips.rds")
```

Biased Random Walks

Define parameters of logistic function to bias transition probabilities

```
diffusion.logistic <- pseudotimeDetermineLogistic(object, "pseudotime", optimal.cells.forward = 40,
    max.cells.back = 80, pseudotime.direction = "<", do.plot = T, print.values = T)
```




```
## [1] "Mean pseudotime back (~80 cells) 0.00138123109435677"
## [1] "Chance of accepted move to equal pseudotime is 0.81741461840178"
## [1] "Mean pseudotime forward (~40 cells) -0.000702878135827306"
```

Run walks on cluster

Biased random walks were run on the cluster using the scripts URD-TM.R, URD-TM.sh (to build a biased transition matrix that could be re-used for each tip), and then URD-Walks.R and URD-Walks.sh to parallelize the process of walking from each tip. The commands run by the scripts (if you have a smaller data set that could run on a laptop, for instance) were:

```
# Create biased transition matrix
biased.tm <- pseudotimeWeightTransitionMatrix(object, pseudotime = "pseudotime",
    logistic.params = diffusion.logistic, pseudotime.direction = "<")
```

```
# Define the root cells
root.cells <- rownames(object@meta)[object@meta$STAGE == "ZFHIGH"]

# Define the tip cells
tips <- setdiff(unique(object@group.ids[, clustering]), NA)
this.tip <- tips[tip.to.walk]  # tip.to.walk was passed by the cluster job array.
tip.cells <- rownames(object@group.ids)[which(object@group.ids[, clustering] == this.tip)]

# Do the random walks
these.walks <- simulateRandomWalk(start.cells = tip.cells, transition.matrix = biased.tm,
    end.cells = root.cells, n = walks.to.do, end.visits = 1, verbose.freq = round(walks.to.do/20),
    max.steps = 5000)
```

Process walks from cluster

We then load the pre-run walks from the cluster, and process them to determine the visitation frequency of each cell by the walks from each tip. This determines the developmental trajectories, and will be used to determine the branching structure in the data.

```
# Get list of walk files
tip.walk.files <- list.files(path = "walks/dm-8-tm-40-80/", pattern = ".rds", full.names = T)

# Which tips were walked?
tips.walked <- setdiff(unique(object@group.ids$`ZF6S-Cluster-Num`), NA)

# Run through each tip, load the walks, and process them into visitation
# frequency for that tip.
for (tip in tips.walked) {
    # Get the files for that tip
    tip.files <- grep(paste0("walks-", tip, "-"), tip.walk.files, value = T)
    if (length(tip.files) > 0) {
        # Read the files into a list of lists, and do a non-recursive unlist to combine
        # into one list.
        these.walks <- unlist(lapply(tip.files, readRDS), recursive = F)
        object <<- processRandomWalks(object, walks = these.walks, walks.name = tip,
            verbose = F)
    }
}
```

Save objects

```
saveRDS(object, file = "obj/object_5_withWalks.rds")
```

URD 5: Build Tree

```
library(URD)
library(rgl)

# Set up knitr to capture rgl output
rgl::setupKnitr()
```

Load previous saved object

```
object <- readRDS("obj/object_5_withWalks.rds")
```

Refine the walks before building the tree

A few clusters that were run as separate tips are totally intermixed in the diffusion map. In these cases, it's often best to combine the two tips before starting to build the tree. (This averages their visitation frequency, according to the number of cells in each tip, and avoids having to re-run the random walks.)

```
# Load tip cells
object <- loadTipCells(object, tips = "ZF6S-Cluster-Num")

# Combine a few sets of tips where you walked from two groups of cells that
# probably should be considered one, based on the fact that they are intermixed
# in the diffusion map.

# Diencephalon
object <- combineTipVisitation(object, "14", "27", "14")
object <- combineTipVisitation(object, "14", "19", "19")

# Optic Cup
object <- combineTipVisitation(object, "2", "9", "2")

# Combine epidermis and integument
object <- combineTipVisitation(object, "1", "36", "1")

# Tailbud
object <- combineTipVisitation(object, "3", "13", "3")
```

Build the tree

Additionally, only "tips" that are actually terminal populations should be used in construction of the tree if possible. We ran random walks from all 6-somite clusters, and so we exclude several of them here, based on prior knowledge or based on their position in the diffusion map.

```
# Decide on the tips to use in the tree construction
tips.to.exclude <- c("4", "6", "7", "9", "11", "13", "14", "15", "20", "23", "24",
    "25", "27", "28", "30", "35", "36", "37", "39", "41", "42", "44")
tips.to.use <- setdiff(as.character(1:53), tips.to.exclude)
```

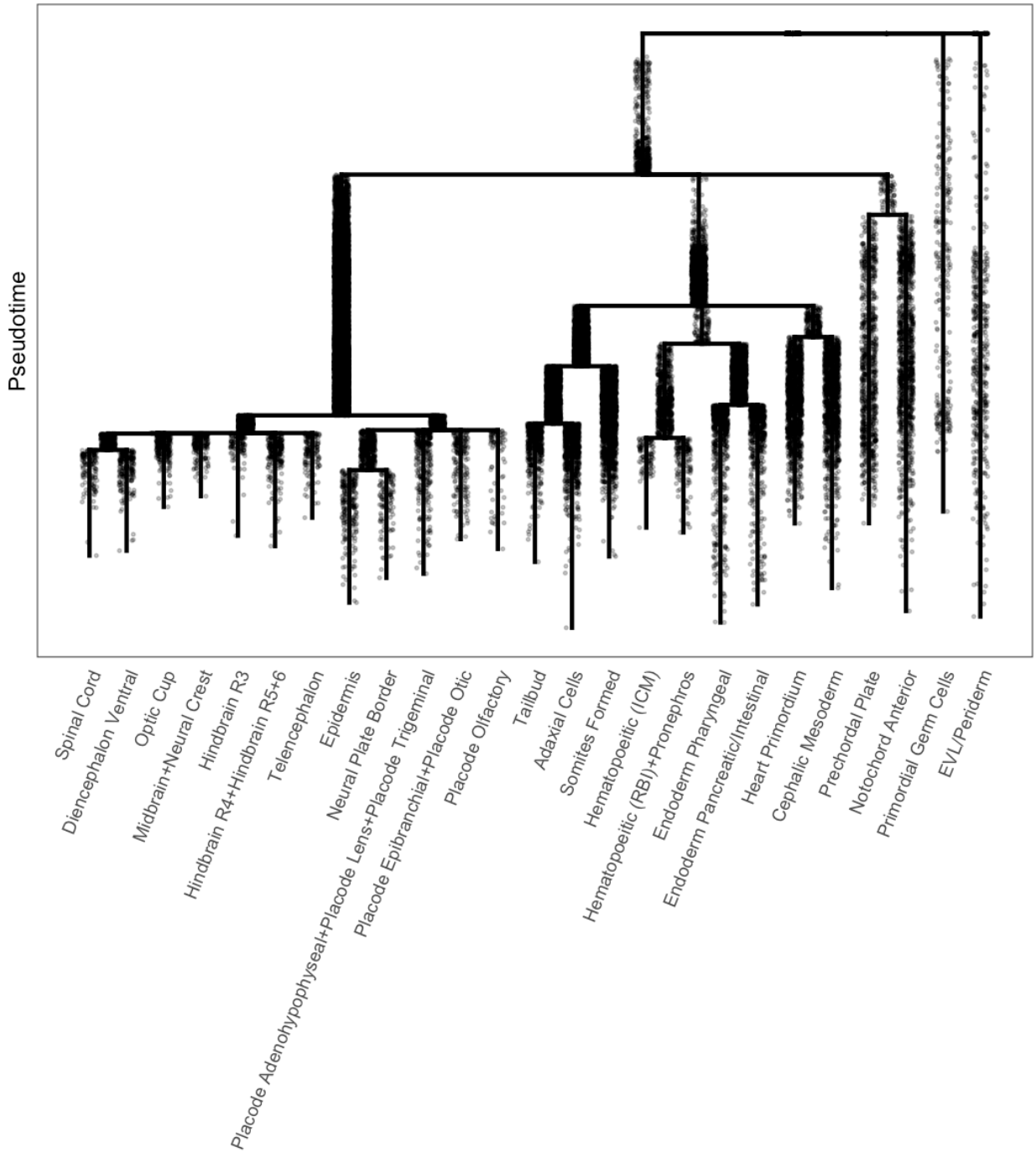
```
# Build the tree
object.built <- buildTree(object = object, pseudotime = "pseudotime", divergence.method = "ks",
    tips.use = tips.to.use, weighted.fusion = T, use.only.original.tips = T, cells.per.pseudotime.bin = 80,
    bins.per.pseudotime.window = 5, minimum.visits = 1, visit.threshold = 0.7, p.thresh = 0.025,
    save.breakpoint.plots = NULL, dendro.node.size = 100, min.cells.per.segment = 10,
    min.pseudotime.per.segment = 0.01, verbose = F)
```

Name the tips

Automated first pass

First, just use the names of the clusters to inspect the tree structure.

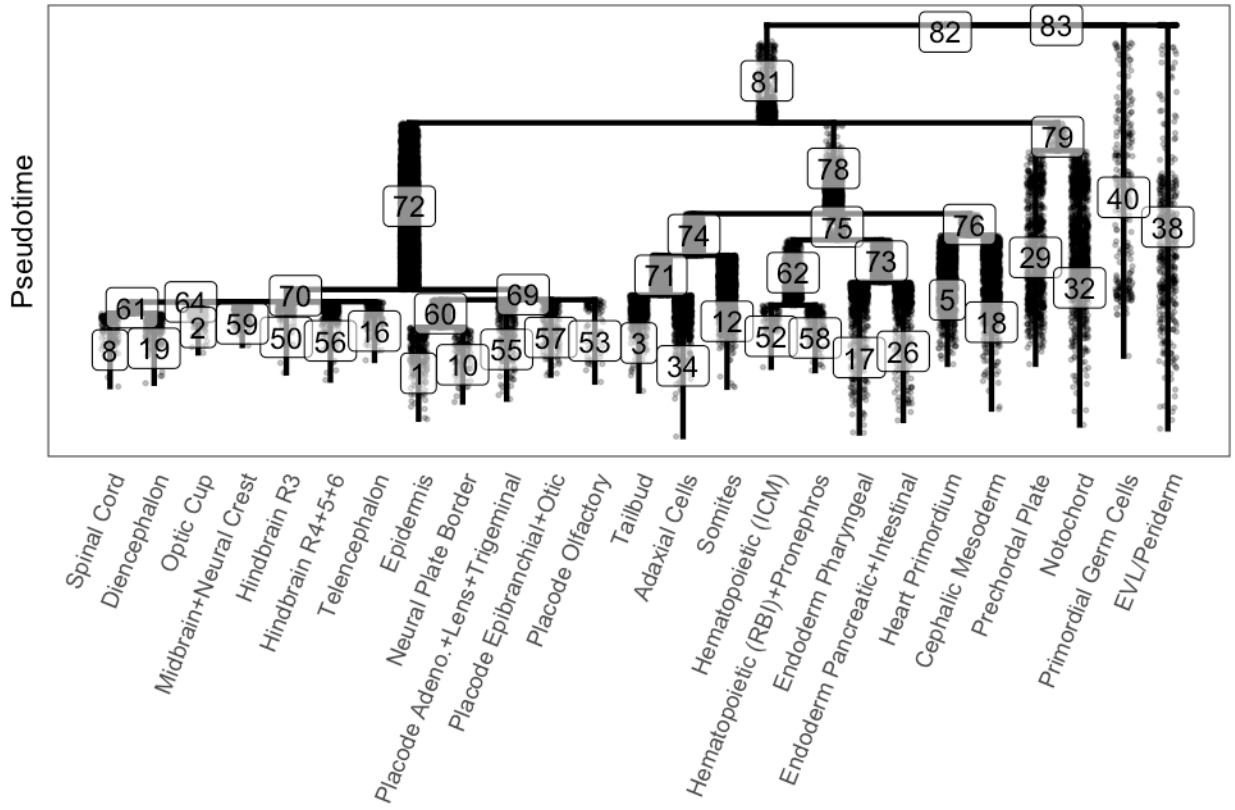
```
tip.names <- unique(object@group.ids[, c("ZF6S-Cluster", "ZF6S-Cluster-Num")])
tip.names <- tip.names[complete.cases(tip.names), ]
object.built <- nameSegments(object.built, segments = tip.names$`ZF6S-Cluster-Num`,
    segment.names = tip.names$`ZF6S-Cluster`)
plotTree(object.built)
```



Manual refinement

And then after inspecting the tree, choose a set of refined names to use going forward, including short names that will look better in the force-directed layout.

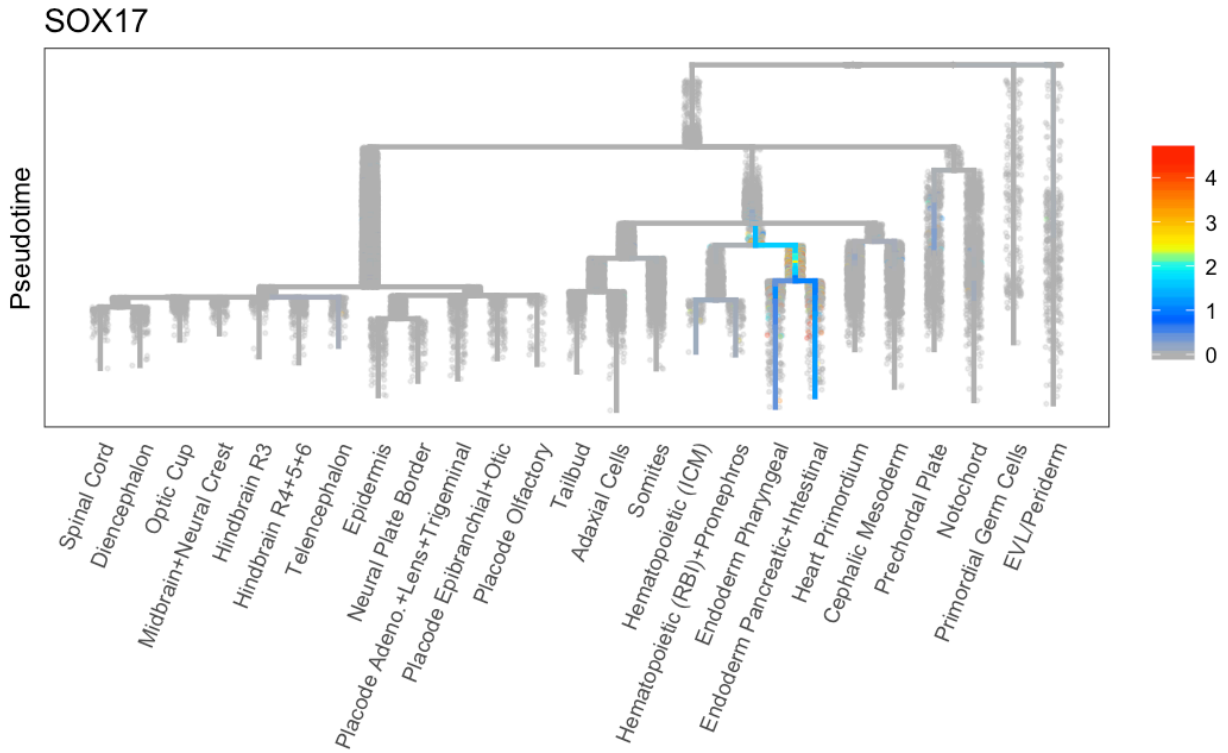
```
# Descriptive names that will be used on dendrogram
new.seg.names <- c("Spinal Cord", "Diencephalon", "Optic Cup", "Midbrain+Neural Crest",
    "Hindbrain R3", "Hindbrain R4+5+6", "Telencephalon", "Epidermis", "Neural Plate Border",
    "Placode Adeno.+Lens+Trigeminal", "Placode Epibranchial+Otic", "Placode Olfactory",
    "Tailbud", "Adaxial Cells", "Somites", "Hematopoietic (ICM)", "Hematopoietic (RBI)+Pronephros",
    "Endoderm Pharyngeal", "Endoderm Pancreatic+Intestinal", "Heart Primordium",
    "Cephalic Mesoderm", "Prechordal Plate", "Notochord", "Primordial Germ Cells",
    "EVL/Periderm")

# Short names / Abbreviations for use on force-directed layout
new.short.names <- c("SC", "Di", "Optic", "MB+NC", "HB R3", "HB R4-6", "Tel", "Epi",
    "NPB", "P(A+L+T)", "P(E+Ot)", "P(Olf)", "TB", "Adax", "Som", "Hem(ICM)", "Hem(RBI)+Pro",
    "Endo Phar", "Endo Pan+Int", "Heart", "CM", "PCP", "Noto", "PGC", "EVL")

# Segment numbers
segs.to.name <- c("8", "19", "2", "59", "50", "56", "16", "1", "10", "55", "57",
    "53", "3", "34", "12", "52", "58", "17", "26", "5", "18", "29", "32", "40", "38")

# Run the naming
object.built <- nameSegments(object.built, segments = segs.to.name, segment.names = new.seg.names,
    short.names = new.short.names)
plotTree(object.built, label.segments = T)
```

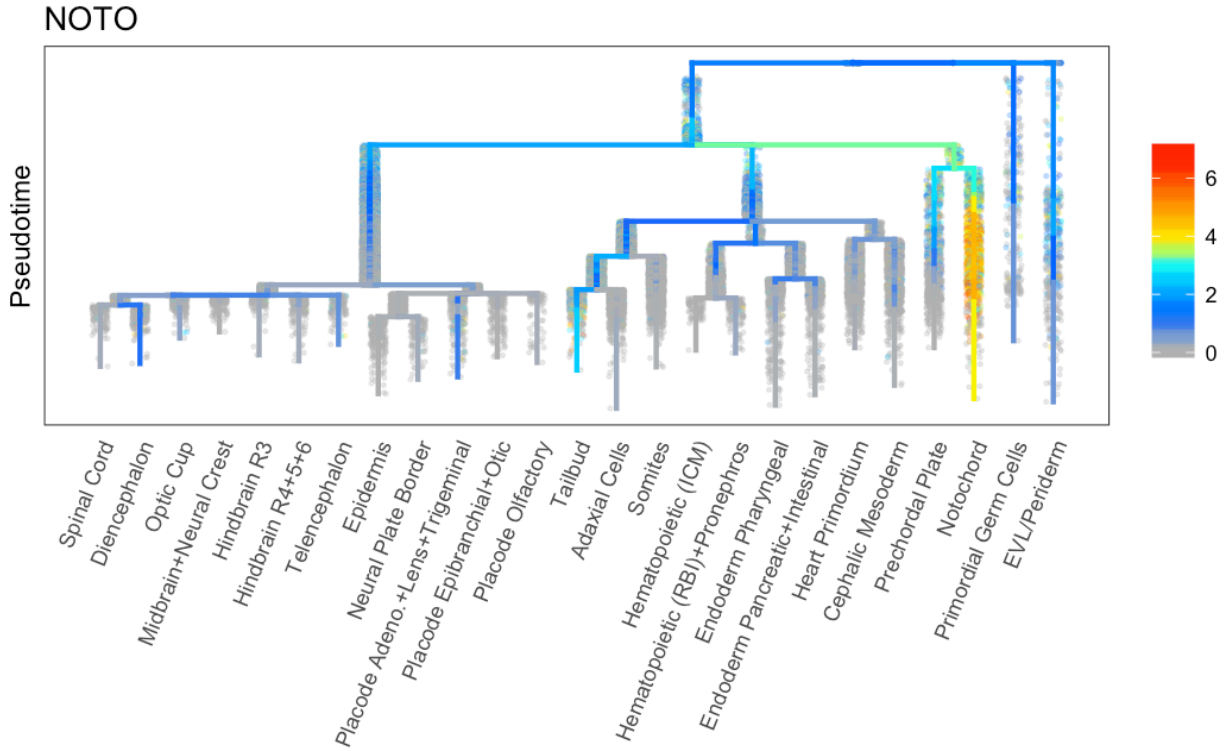


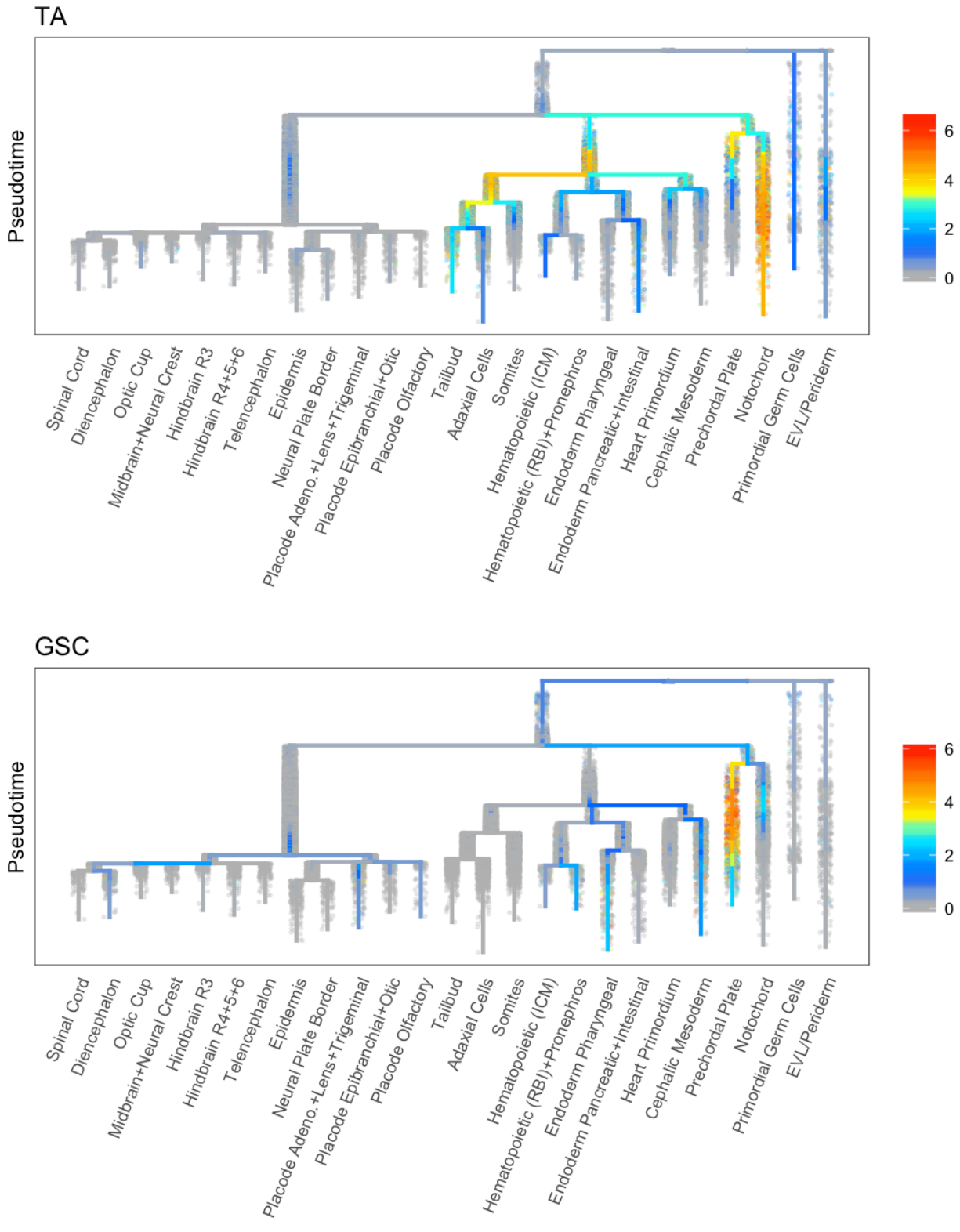
Check out gene expression in dendrogram

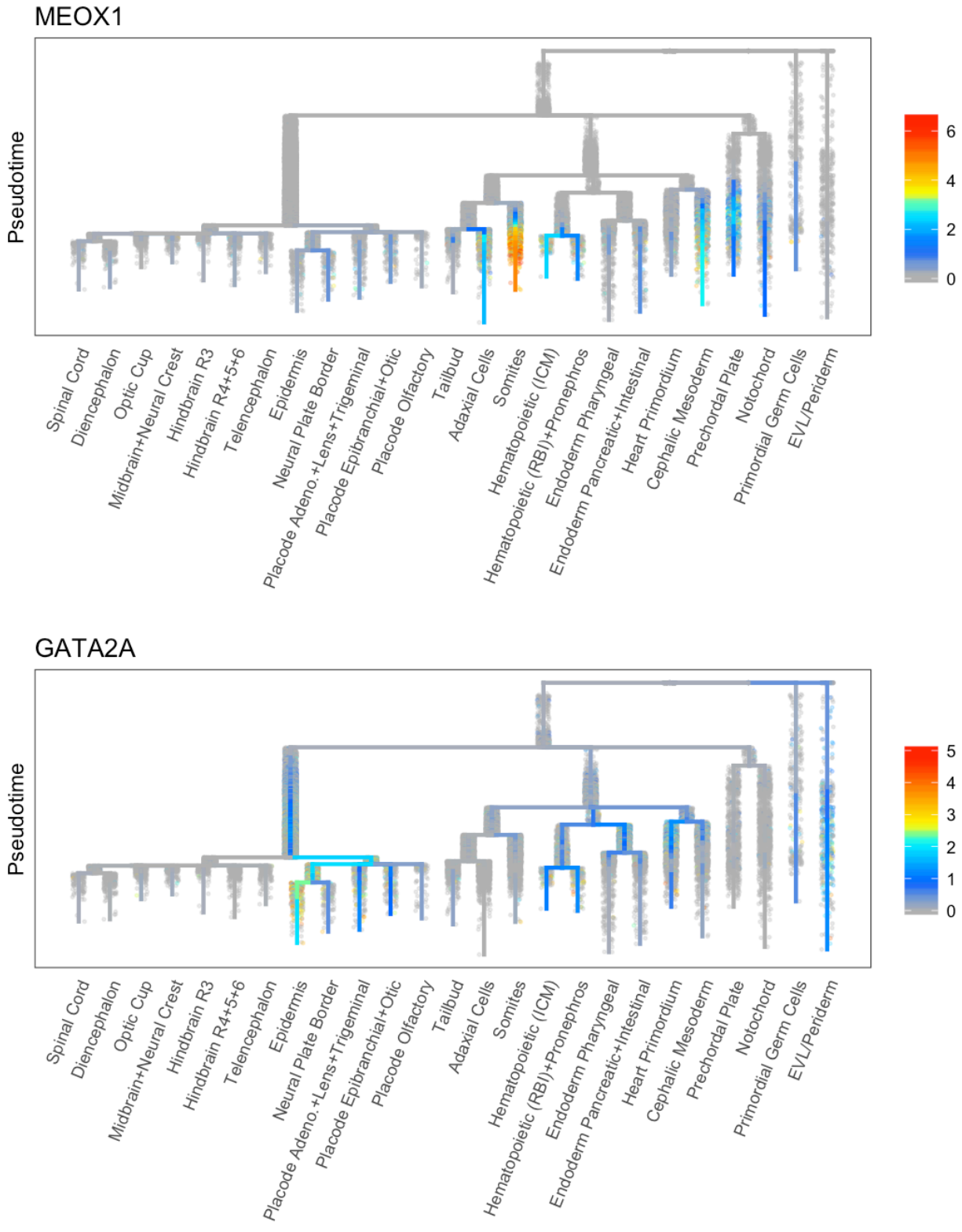
```
genes.plot <- c("SOX17", "NOTO", "TA", "GSC", "MEOX1", "GATA2A", "NANOS3", "KRT4",
    "FSTA", "WNT8A", "CRABP2A", "EGR2B", "ENG2B", "TAL1", "MAFBA", "EMX3")
```

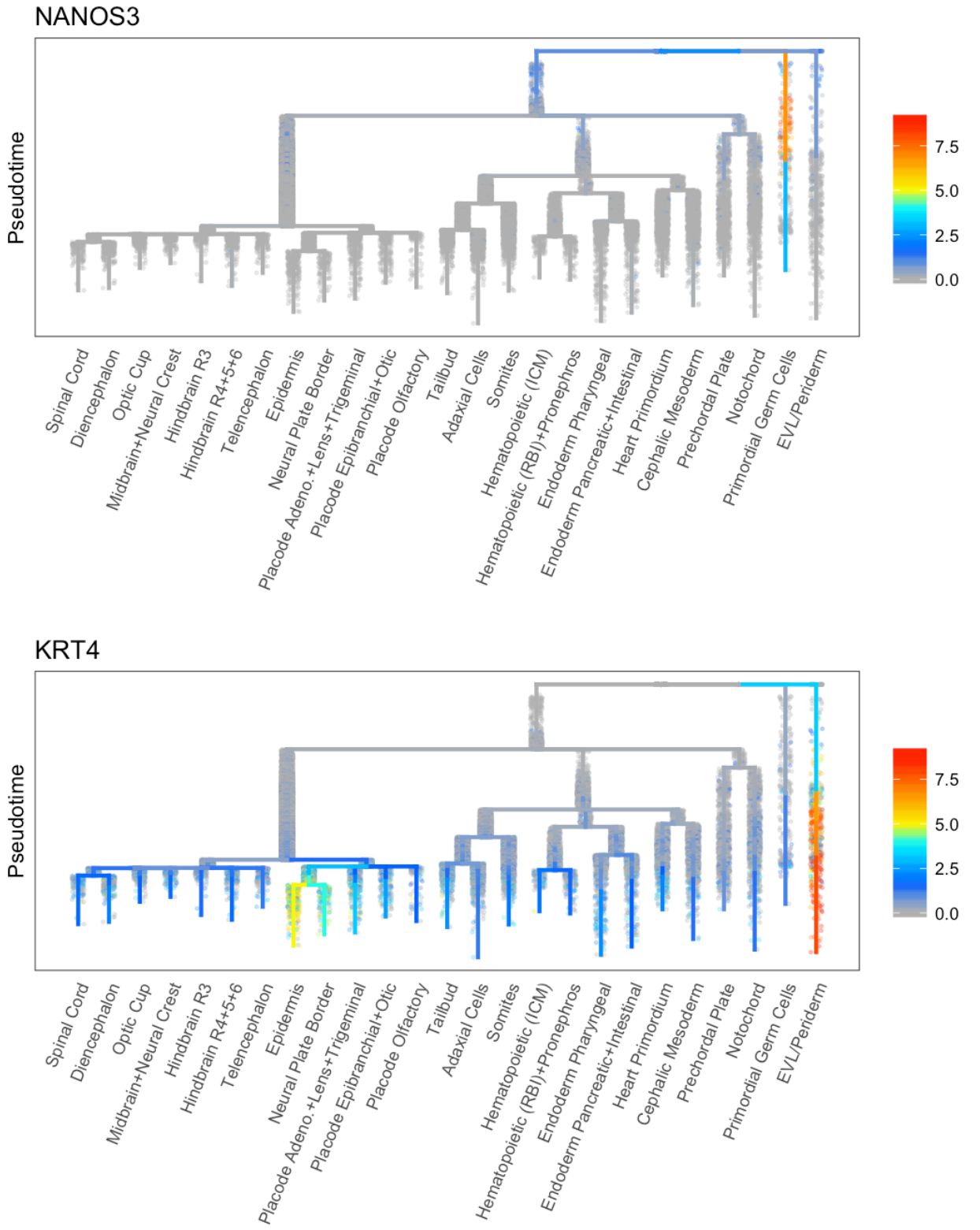
```
for (gene in genes.plot) {
    plot(plotTree(object.built, gene))
}
```

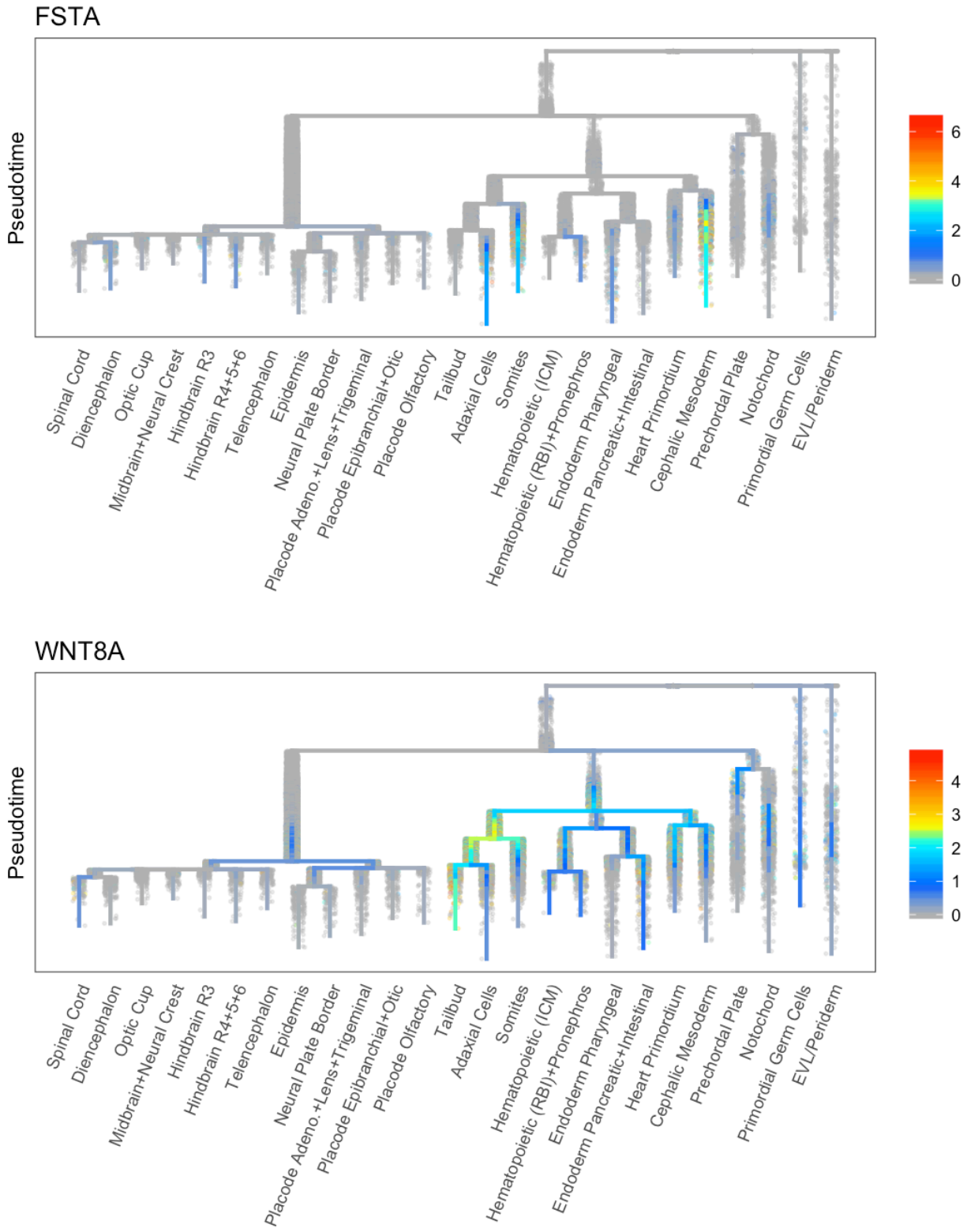


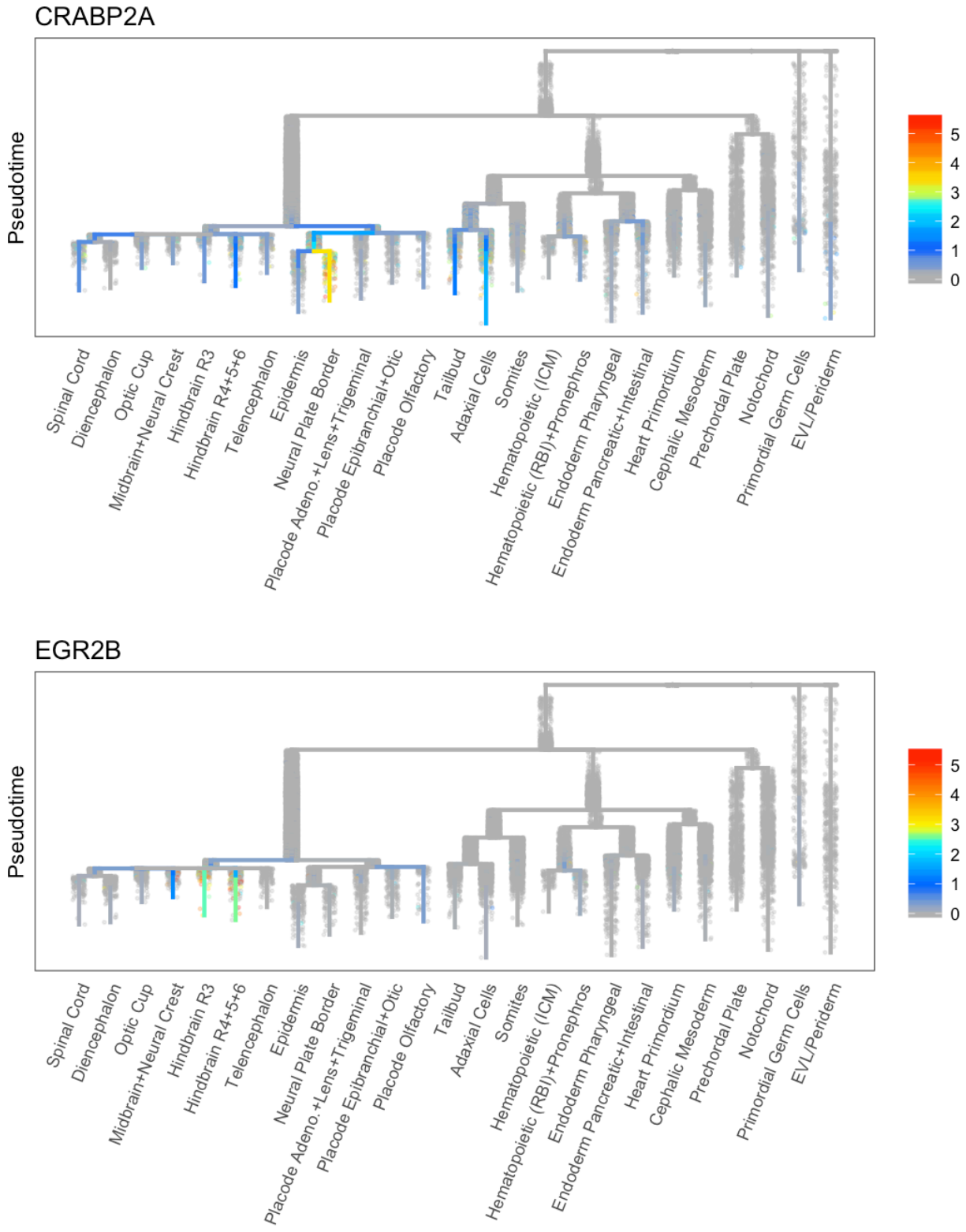


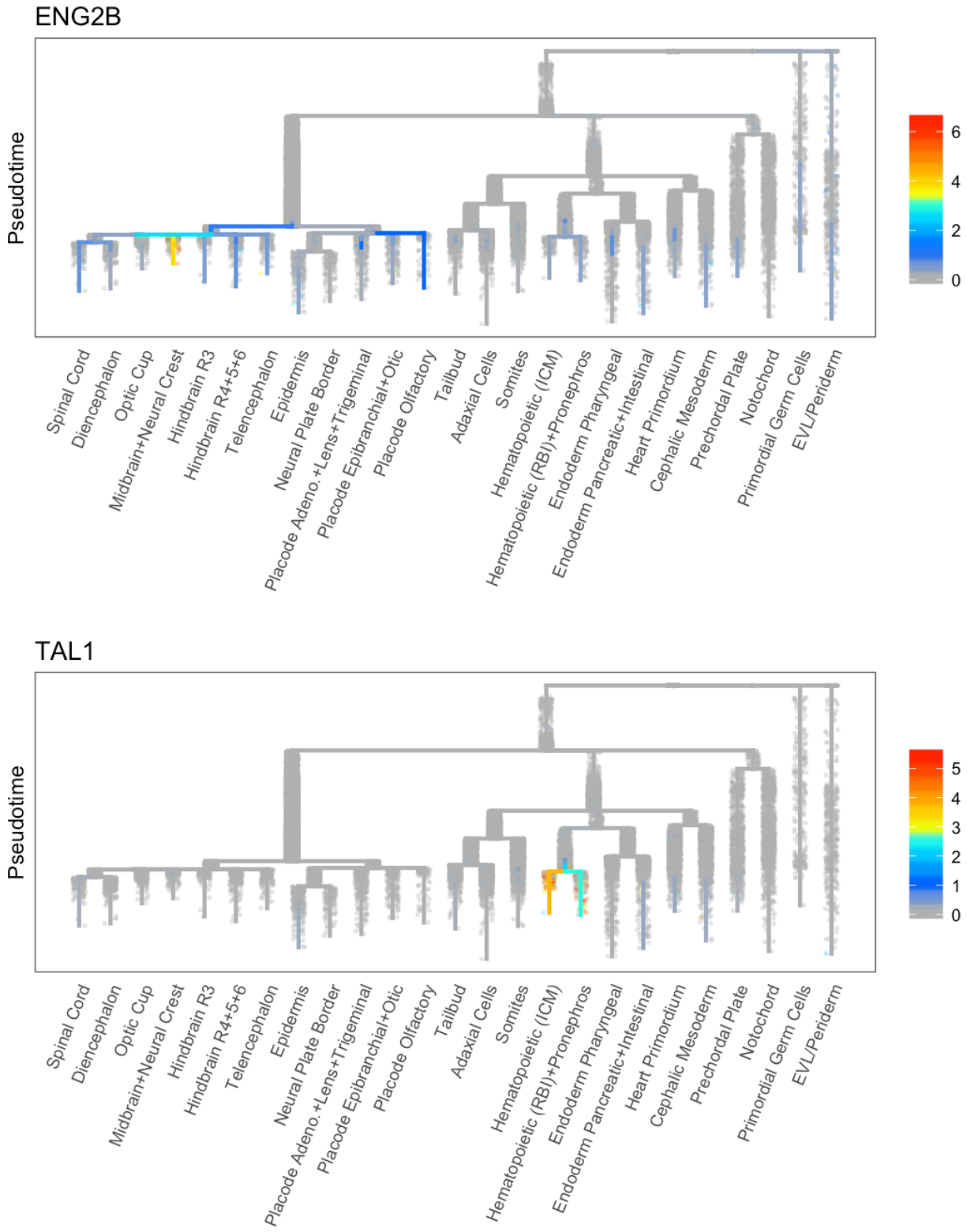

TA
Pseudotime
6
4
2
0
Spinal Cord
Diencephalon
Optic Cup
Midbrain+Neural Crest
Hindbrain R3
Hindbrain R4+5+6
Telencephalon
Epidermis
Neural Plate Border
Placode Adeno.+Lens+Trigeminal
Placode Epibranchial+Otic
Placode Olfactory
Tailbud
Adaxial Cells
Somites
Hematopoietic (ICM)
Hematopoietic (RBI)+Pronephros
Endoderm Pharyngeal
Endoderm Pancreatic+Intestinal
Heart Primordium
Cephalic Mesoderm
Prechordal Plate
Notochord
Primordial Germ Cells
EVL/Periderm
GSC
Pseudotime
6
4
2
0
Spinal Cord
Diencephalon
Optic Cup
Midbrain+Neural Crest
Hindbrain R3
Hindbrain R4+5+6
Telencephalon
Epidermis
Neural Plate Border
Placode Adeno.+Lens+Trigeminal
Placode Epibranchial+Otic
Placode Olfactory
Tailbud
Adaxial Cells
Somites
Hematopoietic (ICM)
Hematopoietic (RBI)+Pronephros
Endoderm Pharyngeal
Endoderm Pancreatic+Intestinal
Heart Primordium
Cephalic Mesoderm
Prechordal Plate
Notochord
Primordial Germ Cells
EVL/Periderm

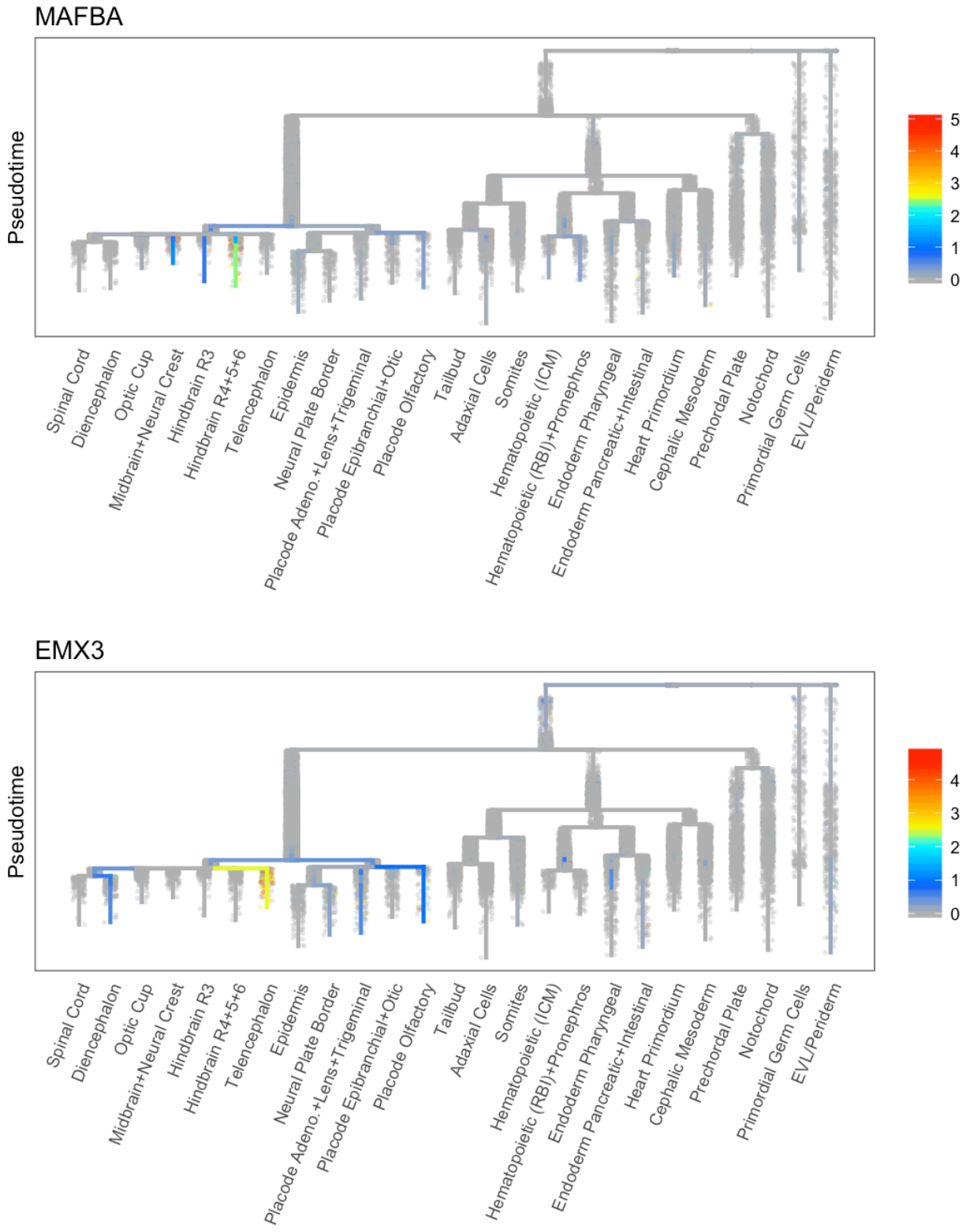

MEOX1
Pseudotime
6
4
2
0
Spinal Cord
Diencephalon
Optic Cup
Midbrain+Neural Crest
Hindbrain R3
Hindbrain R4+5+6
Telencephalon
Epidermis
Neural Plate Border
Placode Adeno.+Lens+Trigeminal
Placode Epibranchial+Otic
Placode Olfactory
Tailbud
Adaxial Cells
Somites
Hematopoietic (ICM)
Hematopoietic (RBI)+Pronephros
Endoderm Pharyngeal
Endoderm Pancreatic+Intestinal
Heart Primordium
Cephalic Mesoderm
Prechordal Plate
Notochord
Primordial Germ Cells
EVL/Periderm
GATA2A
Pseudotime
5
4
3
2
1
0
Spinal Cord
Diencephalon
Optic Cup
Midbrain+Neural Crest
Hindbrain R3
Hindbrain R4+5+6
Telencephalon
Epidermis
Neural Plate Border
Placode Adeno.+Lens+Trigeminal
Placode Epibranchial+Otic
Placode Olfactory
Tailbud
Adaxial Cells
Somites
Hematopoietic (ICM)
Hematopoietic (RBI)+Pronephros
Endoderm Pharyngeal
Endoderm Pancreatic+Intestinal
Heart Primordium
Cephalic Mesoderm
Prechordal Plate
Notochord
Primordial Germ Cells
EVL/Periderm


NANOS3
Pseudotime
7.5
5.0
2.5
0.0
Spinal Cord
Diencephalon
Optic Cup
Midbrain+Neural Crest
Hindbrain R3
Hindbrain R4+5+6
Telencephalon
Epidermis
Neural Plate Border
Placode Adeno.+Lens+Trigeminal
Placode Epibranchial+Otic
Placode Olfactory
Tailbud
Adaxial Cells
Somites
Hematopoietic (ICM)
Hematopoietic (RBI)+Pronephros
Endoderm Pharyngeal
Endoderm Pancreatic+Intestinal
Heart Primordium
Cephalic Mesoderm
Prechordal Plate
Notochord
Primordial Germ Cells
EVL/Periderm
KRT4
Pseudotime
7.5
5.0
2.5
0.0
Spinal Cord
Diencephalon
Optic Cup
Midbrain+Neural Crest
Hindbrain R3
Hindbrain R4+5+6
Telencephalon
Epidermis
Neural Plate Border
Placode Adeno.+Lens+Trigeminal
Placode Epibranchial+Otic
Placode Olfactory
Tailbud
Adaxial Cells
Somites
Hematopoietic (ICM)
Hematopoietic (RBI)+Pronephros
Endoderm Pharyngeal
Endoderm Pancreatic+Intestinal
Heart Primordium
Cephalic Mesoderm
Prechordal Plate
Notochord
Primordial Germ Cells
EVL/Periderm


FSTA
Pseudotime
6
4
2
0
Spinal Cord
Diencephalon
Optic Cup
Midbrain+Neural Crest
Hindbrain R3
Hindbrain R4+5+6
Telencephalon
Epidermis
Neural Plate Border
Placode Adeno.+Lens+Trigeminal
Placode Epibranchial+Otic
Placode Olfactory
Tailbud
Adaxial Cells
Somites
Hematopoietic (ICM)
Hematopoietic (RBI)+Pronephros
Endoderm Pharyngeal
Endoderm Pancreatic+Intestinal
Heart Primordium
Cephalic Mesoderm
Prechordal Plate
Notochord
Primordial Germ Cells
EVL/Periderm
WNT8A
Pseudotime
4
3
2
1
0
Spinal Cord
Diencephalon
Optic Cup
Midbrain+Neural Crest
Hindbrain R3
Hindbrain R4+5+6
Telencephalon
Epidermis
Neural Plate Border
Placode Adeno.+Lens+Trigeminal
Placode Epibranchial+Otic
Placode Olfactory
Tailbud
Adaxial Cells
Somites
Hematopoietic (ICM)
Hematopoietic (RBI)+Pronephros
Endoderm Pharyngeal
Endoderm Pancreatic+Intestinal
Heart Primordium
Cephalic Mesoderm
Prechordal Plate
Notochord
Primordial Germ Cells
EVL/Periderm


CRABP2A
Pseudotime
5
4
3
2
1
0
Spinal Cord
Diencephalon
Optic Cup
Midbrain+Neural Crest
Hindbrain R3
Hindbrain R4+5+6
Telencephalon
Epidermis
Neural Plate Border
Placode Adeno.+Lens+Trigeminal
Placode Epibranchial+Otic
Placode Olfactory
Tailbud
Adaxial Cells
Somites
Hematopoietic (ICM)
Hematopoietic (RBI)+Pronephros
Endoderm Pharyngeal
Endoderm Pancreatic+Intestinal
Heart Primordium
Cephalic Mesoderm
Prechordal Plate
Notochord
Primordial Germ Cells
EVL/Periderm
EGR2B
Pseudotime
5
4
3
2
1
0
Spinal Cord
Diencephalon
Optic Cup
Midbrain+Neural Crest
Hindbrain R3
Hindbrain R4+5+6
Telencephalon
Epidermis
Neural Plate Border
Placode Adeno.+Lens+Trigeminal
Placode Epibranchial+Otic
Placode Olfactory
Tailbud
Adaxial Cells
Somites
Hematopoietic (ICM)
Hematopoietic (RBI)+Pronephros
Endoderm Pharyngeal
Endoderm Pancreatic+Intestinal
Heart Primordium
Cephalic Mesoderm
Prechordal Plate
Notochord
Primordial Germ Cells
EVL/Periderm


ENG2B
Pseudotime
6
4
2
0
Spinal Cord
Diencephalon
Optic Cup
Midbrain+Neural Crest
Hindbrain R3
Hindbrain R4+5+6
Telencephalon
Epidermis
Neural Plate Border
Placode Adeno.+Lens+Trigeminal
Placode Epibranchial+Otic
Placode Olfactory
Tailbud
Adaxial Cells
Somites
Hematopoietic (ICM)
Hematopoietic (RBI)+Pronephros
Endoderm Pharyngeal
Endoderm Pancreatic+Intestinal
Heart Primordium
Cephalic Mesoderm
Prechordal Plate
Notochord
Primordial Germ Cells
EVL/Periderm
TAL1
Pseudotime
5
4
3
2
1
0
Spinal Cord
Diencephalon
Optic Cup
Midbrain+Neural Crest
Hindbrain R3
Hindbrain R4+5+6
Telencephalon
Epidermis
Neural Plate Border
Placode Adeno.+Lens+Trigeminal
Placode Epibranchial+Otic
Placode Olfactory
Tailbud
Adaxial Cells
Somites
Hematopoietic (ICM)
Hematopoietic (RBI)+Pronephros
Endoderm Pharyngeal
Endoderm Pancreatic+Intestinal
Heart Primordium
Cephalic Mesoderm
Prechordal Plate
Notochord
Primordial Germ Cells
EVL/Periderm


MAFBA
Pseudotime
5
4
3
2
1
0
Spinal Cord
Diencephalon
Optic Cup
Midbrain+Neural Crest
Hindbrain R3
Hindbrain R4+5+6
Telencephalon
Epidermis
Neural Plate Border
Placode Adeno.+Lens+Trigeminal
Placode Epibranchial+Otic
Placode Olfactory
Tailbud
Adaxial Cells
Somites
Hematopoietic (ICM)
Hematopoietic (RBI)+Pronephros
Endoderm Pharyngeal
Endoderm Pancreatic+Intestinal
Heart Primordium
Cephalic Mesoderm
Prechordal Plate
Notochord
Primordial Germ Cells
EVL/Periderm
EMX3
Pseudotime
4
3
2
1
0
Spinal Cord
Diencephalon
Optic Cup
Midbrain+Neural Crest
Hindbrain R3
Hindbrain R4+5+6
Telencephalon
Epidermis
Neural Plate Border
Placode Adeno.+Lens+Trigeminal
Placode Epibranchial+Otic
Placode Olfactory
Tailbud
Adaxial Cells
Somites
Hematopoietic (ICM)
Hematopoietic (RBI)+Pronephros
Endoderm Pharyngeal
Endoderm Pancreatic+Intestinal
Heart Primordium
Cephalic Mesoderm
Prechordal Plate
Notochord
Primordial Germ Cells
EVL/Periderm

Generate a force-directed layout

We generate a force-directed layout as a nice visualization of the data. It is constructed by generating a weighted k-nearest neighbor network, based on euclidean distance in visitation space (using the frequency of visitation of each cell by the walks from each tip). The nearest neighbor network is optionally refined based on cells' distance in the dendrogram (in terms of which segments are connected). Cells then push and pull against their neighbors, as the amount of freedom they have to move is slowly decreased until cells are locked into place. This produces a two-dimensional layout, and we then add pseudotime as a third dimension.

Choose cells that were visited more robustly

We find that the force directed layout works best if the most poorly visited (and thus likely poorly connected) cells are excluded.

```
# Data frame to measure cell visitation
visitation <- data.frame(cell = rownames(object.built@diff.data), seg = object.built@diff.data$segment,
    stringsAsFactors = F, row.names = rownames(object.built@diff.data))
visitation$visit <- log10(apply(visitation, 1, function(cr) object.built@diff.data[as.character(cr["cell"]),
    paste0("visitfreq.raw.", as.character(cr["seg"]))]) + 1)

# Choose those cells that were well visited
robustly.visited.cells <- visitation[visitation$visit >= 0.5, "cell"]

# Since some tips of the tree were combined in their entirety, get the terminal
# segments to use as the tips of the force-directed layout.
final.tips <- segTerminal(object.built)
```

Calculate layout

It can be important to try several sets of parameters here. Varying the number of nearest neighbors (num.nn) and the amount of refinement based on the dendrogram (cut.unconnected.segments) affects the layout significantly.

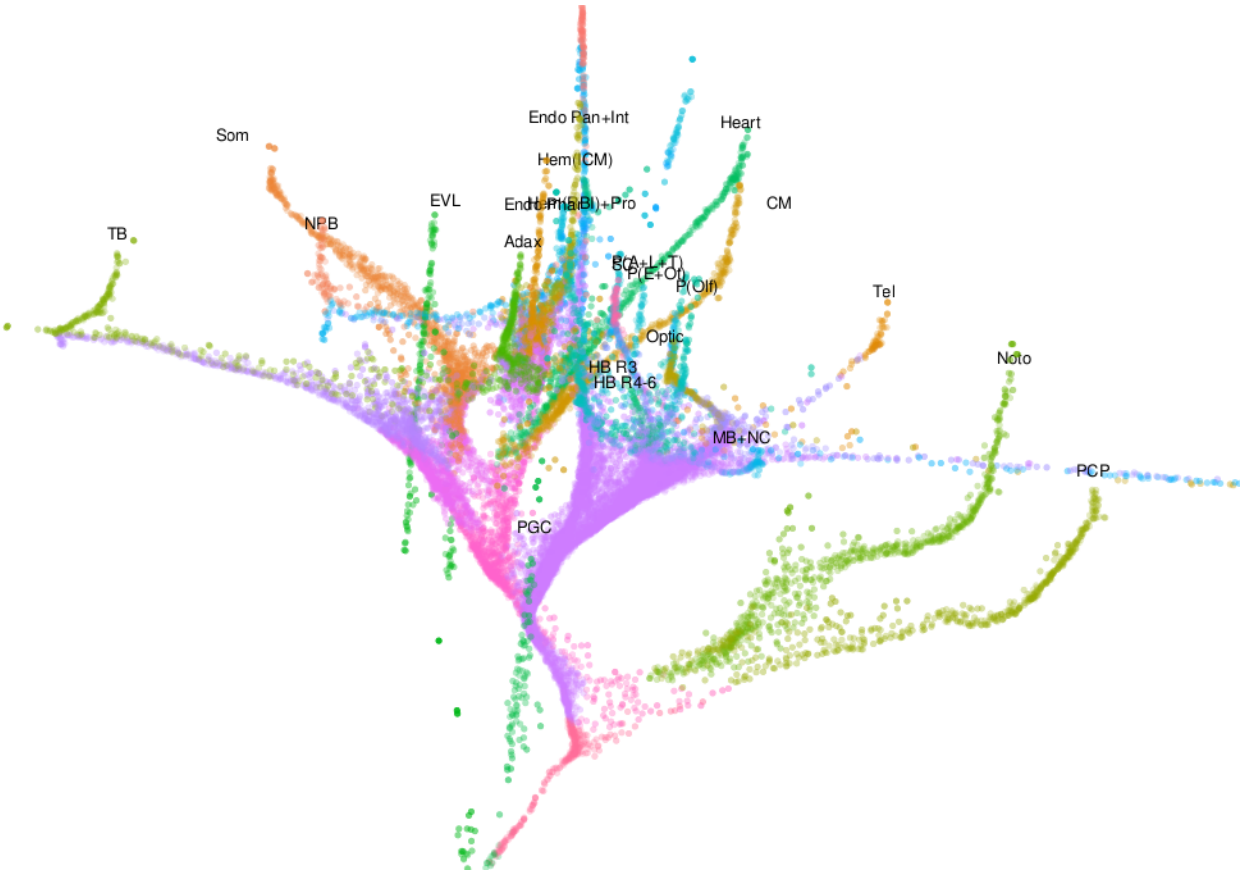
```
# Generate the force-directed layout
object.built <- treeForceDirectedLayout(object.built, num.nn = 120, pseudotime = "pseudotime",
    method = "fr", dim = 2, cells.to.do = robustly.visited.cells, tips = final.tips,
    cut.unconnected.segments = 2, min.final.neighbors = 4, verbose = F)
```

Once calculated and plotted, the plot can be rotated to a desired view, and you can save the view using the function plotTreeForceStore3DView. Thus, many plots with a comparable orientation can then be produced.

Load saved layout

Since the force-directed layout is not deterministic, we instead load a saved layout and orientation to finish the tutorial.

```
# Load view
object.built@tree$force.view.list <- readRDS("fdls/force.view.list.rds")
object.built@tree$force.view.default <- "figure1"

# Load layout
precalc.fdl <- readRDS("fdls/layout.51.cells05.nn120.mn4.cut2.rds")
object.built@tree$walks.force.layout <- precalc.fdl$walks.force.layout

plotTreeForce(object.built, "segment", alpha = 0.2, view = "figure1")
```

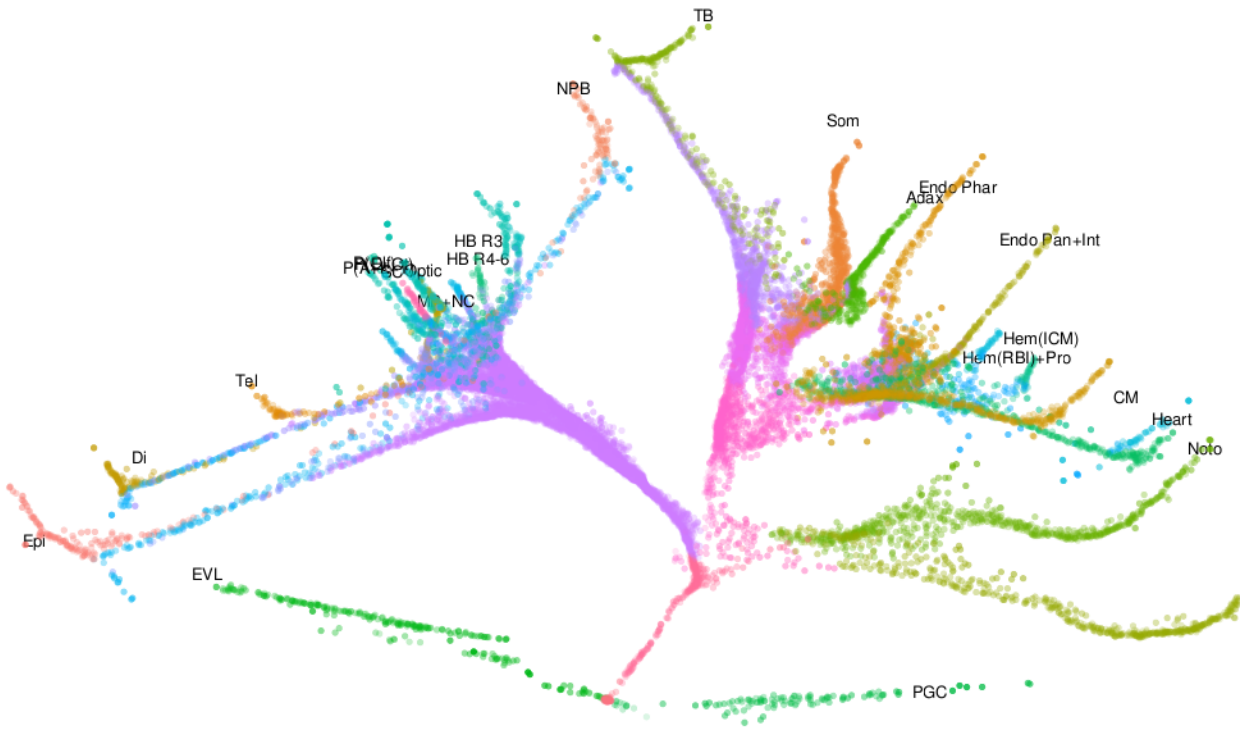


Hand-tune the tree

For optimal 2D presentation in the paper, we further tuned the layout by hand to reduce overlaps and ensure as much of the tree is visible simultaneously as possible. This was done by increasing the angular distance at the first branchpoint, and moving the two completely disconnected populations (EVL & PGCs).

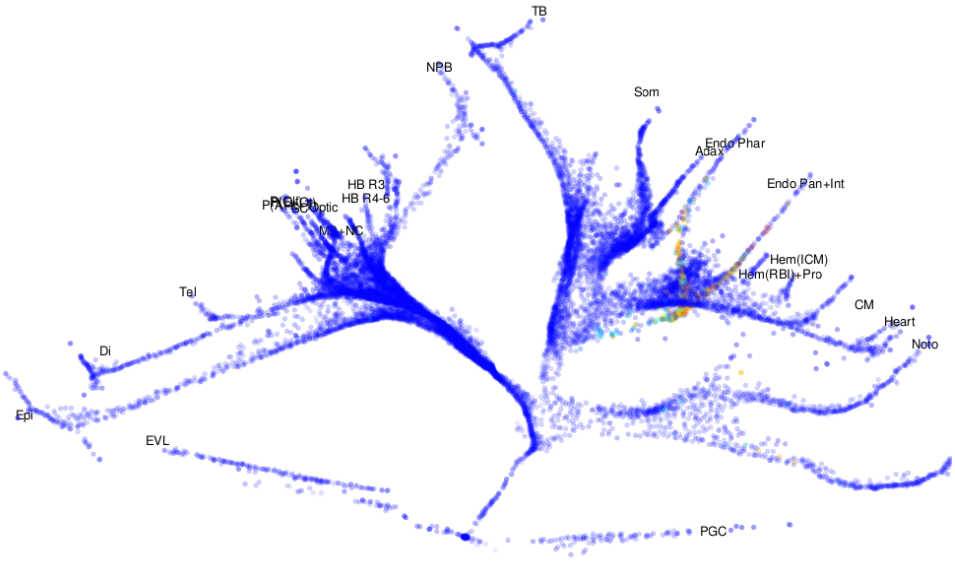
```
## ECTODERM

# Rotate the ectoderm branch around the z-axis to optimize the orientation of the
# neural and non-neural ectoderm later.
object.built <- treeForceRotateCoords(object.built, seg = "72", angle = -3.3, axis = "z",
    around.cell = 10, throw.out.cells = 1000, pseudotime = "pseudotime")

# Curve the ectoderm outwards and forwards, to prevent it overlapping the
# mesoderm, so that the branching structure within the two domains is easily
# viewed.
for (throw.out in c(0, 100, 250, 500, 1000)) {
    object.built <- treeForceRotateCoords(object.built, seg = "72", angle = pi/20,
        axis = "y", around.cell = 10, throw.out.cells = throw.out, pseudotime = "pseudotime")
}
for (throw.out in c(0, 100, 250, 500, 1000)) {
    object.built <- treeForceRotateCoords(object.built, seg = "72", angle = pi/24,
        axis = "z", around.cell = 10, throw.out.cells = throw.out, pseudotime = "pseudotime")
}

## AXIAL MESODERM

# Rotate the axial mesoderm a bit to the right to make some extra space for the
# remainder of the mesendoderm.

object.built <- treeForceRotateCoords(object.built, seg = "79", angle = -pi/10, axis = "y",
    around.cell = 10, throw.out.cells = 0, pseudotime = "pseudotime")
object.built <- treeForceRotateCoords(object.built, seg = "79", angle = -pi/8, axis = "x",
    around.cell = 10, throw.out.cells = 0, pseudotime = "pseudotime")
```

```
## REMAINDER OF THE MESENDODERM

# Rotate the rest of the mesoderm a bit to the right into the empty space between
# the ectoderm and axial mesoderm.

object.built <- treeForceRotateCoords(object.built, seg = "78", angle = pi/4, axis = "z",
    around.cell = 10, throw.out.cells = 0, pseudotime = "pseudotime")
object.built <- treeForceRotateCoords(object.built, seg = "78", angle = -pi/5, axis = "y",
    around.cell = 10, throw.out.cells = 0, pseudotime = "pseudotime")

## PGCs

# The PGCs are disconnected from the tree totally, so just rotate and move them
# into place.

object.built <- treeForceRotateCoords(object.built, seg = "40", angle = -pi/2, axis = "y",
    around.cell = 1, throw.out.cells = 0, pseudotime = "pseudotime")
object.built <- treeForceTranslateCoords(object.built, seg = "40", x = 0, y = 0,
    z = -3)

## EVL / Periderm

# The EVL/Periderm is also nearly completely disconnected.  Rotate these cells,
# and also close the enormous gap in them somewhat so that they fit neatly next
# to the ectoderm.

# Determine the EVL cells
evl.cells <- intersect(cellsInCluster(object.built, "segment", "38"), rownames(object.built@tree$walks.force.layout))
evl.cells.move.1 <- evl.cells[which(object.built@tree$walks.force.layout[evl.cells,
    "telescope.pt"] > 15)]
evl.cells.move.2 <- evl.cells[which(object.built@tree$walks.force.layout[evl.cells,
    "telescope.pt"] > 30)]

# Close the gaps a bit in this lineage to shorten it so that it doesn't overlap
# with the ectoderm
object.built <- treeForceTranslateCoords(object.built, cells = evl.cells.move.1,
    x = 0, y = 0, z = -10)
object.built <- treeForceTranslateCoords(object.built, cells = evl.cells.move.2,
    x = 0, y = 0, z = -15)

# Rotate the EVL cells in next to the ectoderm
object.built <- treeForceRotateCoords(object.built, seg = "38", angle = 1.35, axis = "y",
    around.cell = 1, throw.out.cells = 0, pseudotime = "pseudotime")
plotTreeForce(object.built, "segment", alpha = 0.2, view = "figure1")
```

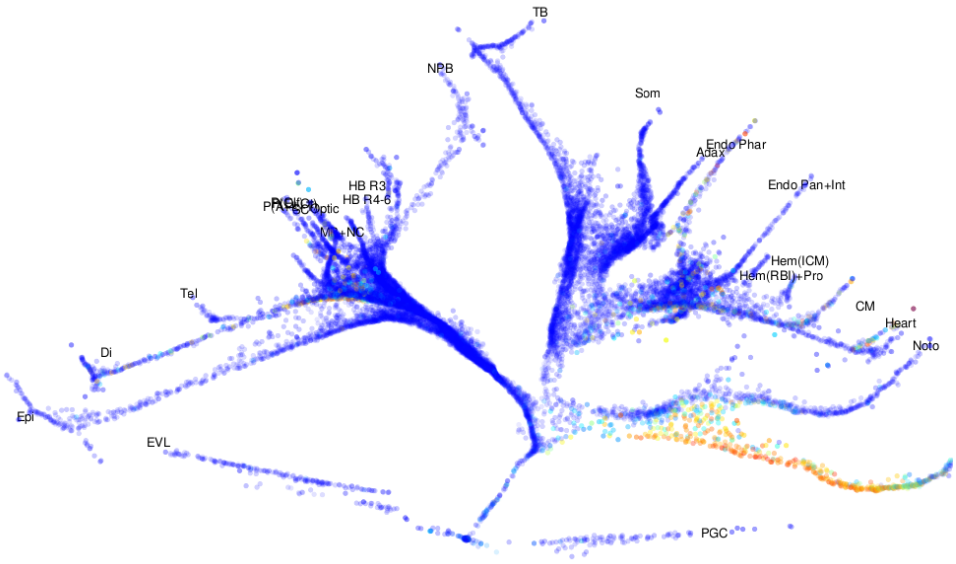


Check out gene expression in the force-directed layout

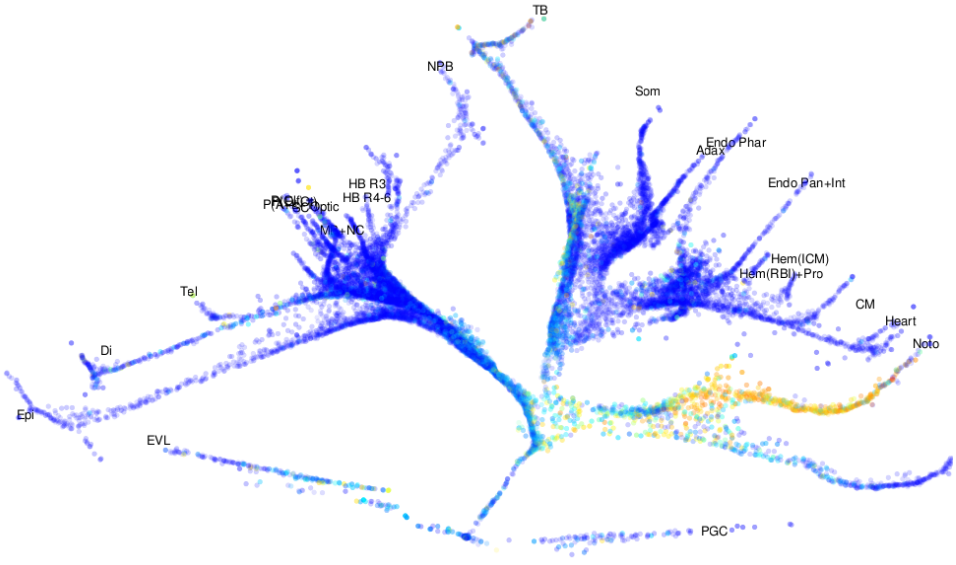
```
plotTreeForce(object.built, "SOX17", alpha = 0.2, view = "figure1")
```



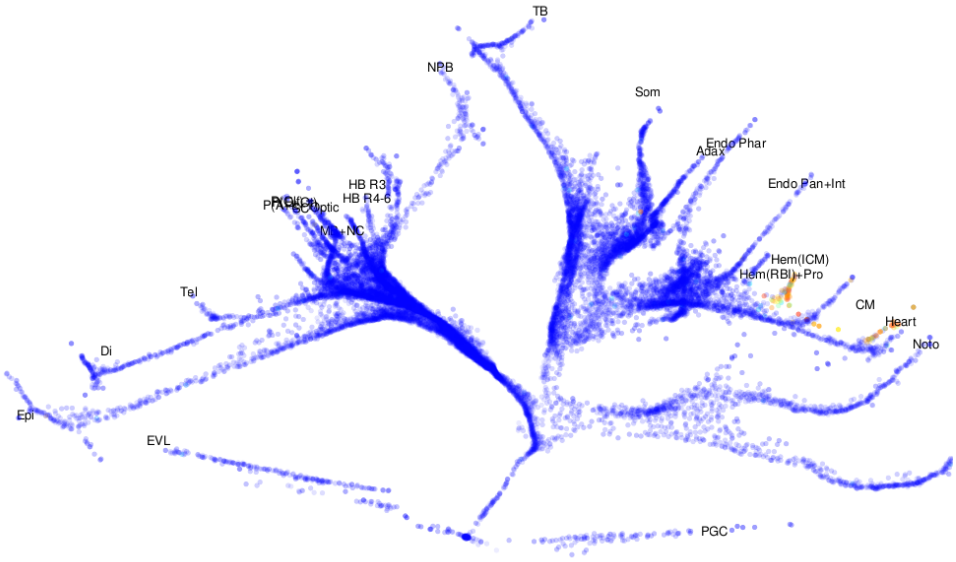
```
plotTreeForce(object.built, "GSC", alpha = 0.2, view = "figure1")
```



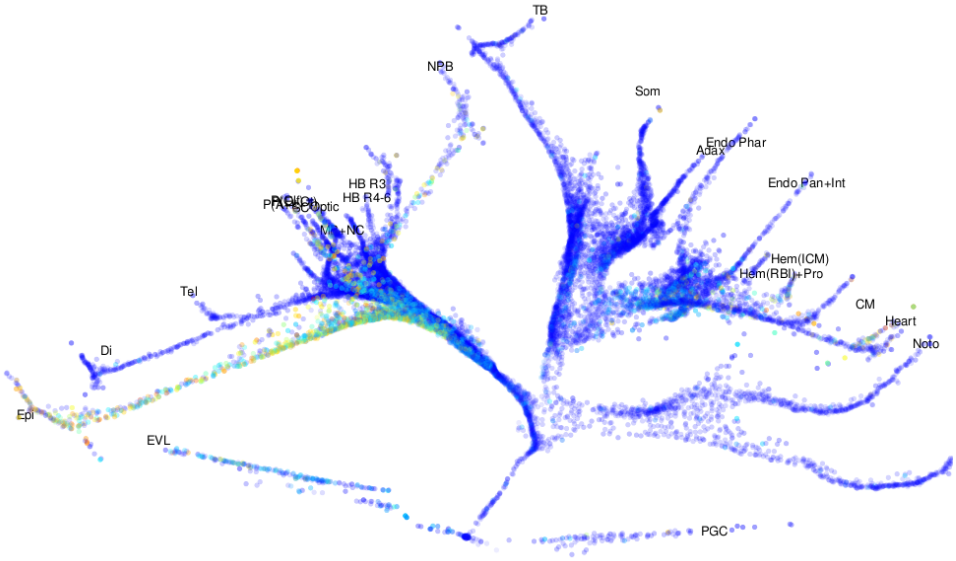

```
plotTreeForce(object.built, "NOTO", alpha = 0.2, view = "figure1")
```



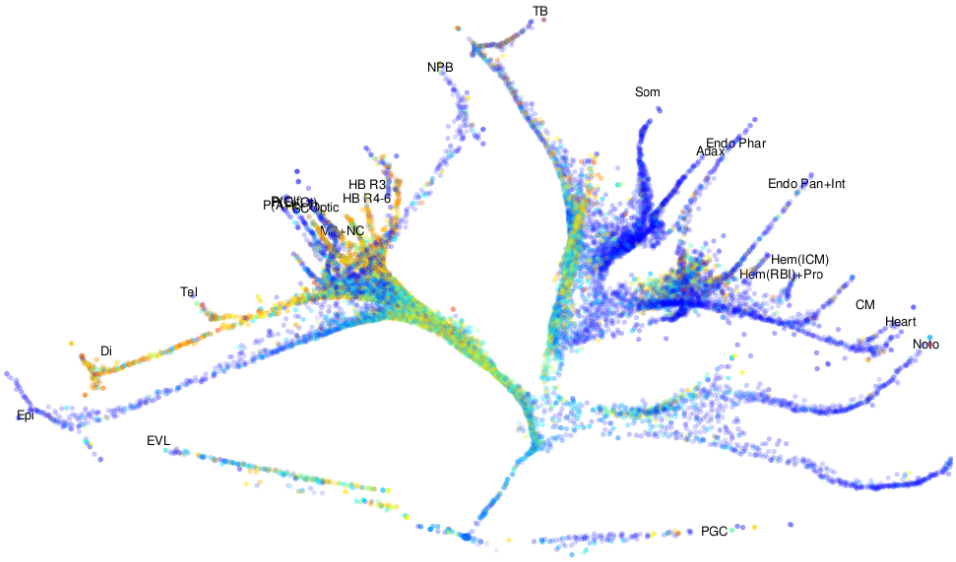

```
plotTreeForce(object.built, "TAL1", alpha = 0.2, view = "figure1")
```




```
plotTreeForce(object.built, "GATA2A", alpha = 0.2, view = "figure1")
```




```
plotTreeForce(object.built, "SOX19A", alpha = 0.2, view = "figure1")
```



Save objects

```
saveRDS(object.built, file = "obj/object_6_tree.rds")
```

URD 6: Gene Expression Cascades

```
library(URD)
```

Load previous saved object

```
object <- readRDS("obj/object_6_tree.rds")
```

Differential expression with precision-recall along URD dendrogram

For each population, we worked backward along that population's trajectory, starting at the tip. We compared cells in each segment pairwise with cells from each of that segment's siblings and children (cropped to the same pseudotime limits as the segment under consideration). Genes were considered differentially expressed if they were expressed in at least 10% of cells in the trajectory segment under consideration, their mean expression was upregulated 1.5x compared to the sibling and the gene was 1.25x better than a random classifier for the population as determined by the area under a precision-recall curve. Genes were considered part of a population's cascade if, at any given branchpoint, they were considered differential against at least 60% of their siblings, and they were not differentially upregulated in a different trajectory downstream of the branchpoint (i.e. upregulated in a shared segment, but really a better marker of a different population).

Precision-recall tests along tree

We performed tests along the tree for all blastoderm trajectories. Segment 64 was skipped, as it contained only 25 cells, and was too sensitive to random variations in expression level.

```
# Determine tips to run DE for
tips.to.run <- setdiff(as.character(object@tree$segment.names), c("Primoridal Germ Cells",
    "EVL/Periderm"))
genes.use <- NULL  # Calculate for all genes

# Calculate the markers of each other population.
gene.markers <- list()
for (tipn in 1:length(tips.to.run)) {
    tip <- tips.to.run[tipn]
    print(paste0(Sys.time(), ": ", tip))
    markers <- aucprTestAlongTree(object, pseudotime = "pseudotime", tips = tip,
        log.effect.size = 0.4, auc.factor = 1.25, max.auc.threshold = 0.85, frac.must.express = 0.1,
        frac.min.diff = 0, genes.use = genes.use, root = "81", only.return.global = F,
        must.beat.sibs = 0.6, report.debug = T, segs.to.skip = "64")
    saveRDS(markers, file = paste0("cascades/aucpr/", tip, ".rds"))
    gene.markers[[tip]] <- markers
}
```

```
## Warning in as.POSIXlt.POSIXct(x, tz): unknown timezone 'zone/tz/2017c.1.0/
## zoneinfo/America/New_York'
```

```
## [1] "2018-02-24 16:40:27: Spinal Cord"
## [1] "2018-02-24 16:53:48: Diencephalon"
## [1] "2018-02-24 17:05:59: Optic Cup"
## [1] "2018-02-24 17:15:51: Midbrain+Neural Crest"
## [1] "2018-02-24 17:25:31: Hindbrain R3"
## [1] "2018-02-24 17:36:24: Hindbrain R4+5+6"
## [1] "2018-02-24 17:46:17: Telencephalon"
## [1] "2018-02-24 17:57:23: Epidermis"
## [1] "2018-02-24 18:15:23: Neural Plate Border"
## [1] "2018-02-24 18:33:48: Placode Adeno.+Lens+Trigeminal"
## [1] "2018-02-24 18:46:49: Placode Epibranchial+Otic"
## [1] "2018-02-24 18:59:36: Placode Olfactory"
## [1] "2018-02-24 19:15:32: Tailbud"
## [1] "2018-02-24 19:22:49: Adaxial Cells"
## [1] "2018-02-24 19:29:54: Somites"
## [1] "2018-02-24 19:34:51: Hematopoeitic (ICM)"
## [1] "2018-02-24 19:42:04: Hematopoeitic (RBI)+Pronephros"
## [1] "2018-02-24 19:48:53: Endoderm Pharyngeal"
```

```
## [1] "2018-02-24 19:54:33: Endoderm Pancreatic+Intestinal"
## [1] "2018-02-24 20:00:26: Heart Primordium"
## [1] "2018-02-24 20:05:11: Cephalic Mesoderm"
## [1] "2018-02-24 20:10:07: Prechordal Plate"
## [1] "2018-02-24 20:15:28: Notochord"
## [1] "2018-02-24 20:20:43: Primordial Germ Cells"
```

Segment 75 gave numerous markers that did not appear to be overall markers of the trajectories that pass through segment 75 when plotted on the tree. Thus, we excluded markers that were solely upregulated in segment 75.

```
# Segment 75 was giving very bizarre differential expression results, so we
# removed any markers that were only markers in segment 75.
pops.fix <- c("Endoderm Pharyngeal", "Endoderm Pancreatic+Intestinal", "Hematopoeitic (ICM)",
    "Hematopoeitic (RBI)+Pronephros")
for (pop in pops.fix) {
    mc <- gene.markers[[pop]]$marker.chain
    seg.good <- grep("75", names(mc), value = T, invert = T)
    mark.ok <- unique(unlist(lapply(mc[seg.good], rownames)))
    mark.keep <- intersect(mark.ok, rownames(gene.markers[[pop]]$diff.exp))
    gene.markers[[pop]]$diff.exp <- gene.markers[[pop]]$diff.exp[mark.keep, ]
    saveRDS(gene.markers[[pop]], file = paste0("cascades/aucpr/", pop, ".rds"))
}
```

Precision-recall tests by stage for PGCs and EVL

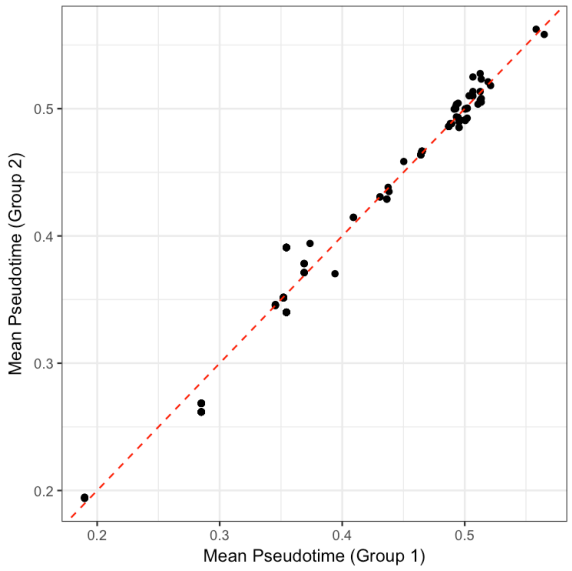
The primordial germ cells (PGCs) and enveloping layer cells (EVL) are distinct from the beginning of our tree. Thus, they are not divided up into small segments, and our differential expression test along the tree performed poorly for them. So, instead, we divided cells into five groups, based on their developmental stage, and performed pairwise comparisons at each stage using the precision-recall approach, and kept those genes that were markers of at least 3 groups.

```
# Define cell populations
evl.cells <- cellsInCluster(object, "segment", "38")
pgc.cells <- cellsInCluster(object, "segment", "40")
blastoderm.cells <- cellsInCluster(object, "segment", segChildrenAll(object, "81",
    include.self = T))

# Copy STAGE to group.ids for the TestByFactor function
object@group.ids$STAGE <- object@meta[rownames(object@group.ids), "STAGE"]

# Define stage groups
groups <- list(c("ZFHIGH", "ZFOBLONG"), c("ZFDOME", "ZF30"), c("ZF50", "ZFS", "ZF60"),
    c("ZF75", "ZF90"), c("ZFB", "ZF3S", "ZF6S"))

# Calculate markers
evl.markers.bystage <- aucprTestByFactor(object, cells.1 = evl.cells, cells.2 = list(pgc.cells,
    blastoderm.cells), label = "STAGE", groups = groups, log.effect.size = 0.5, auc.factor = 1,
    min.auc.thresh = 0.1, max.auc.thresh = Inf, frac.must.express = 0.1, frac.min.diff = 0,
    genes.use = genes.use, min.groups.to.mark = 3, report.debug = T)
pgc.markers.bystage <- aucprTestByFactor(object, cells.1 = pgc.cells, cells.2 = list(evl.cells,
    blastoderm.cells), label = "STAGE", groups = groups, log.effect.size = 0.5, auc.factor = 1,
    min.auc.thresh = 0.1, max.auc.thresh = Inf, frac.must.express = 0.1, frac.min.diff = 0,
    genes.use = genes.use, min.groups.to.mark = 3, report.debug = T)

# Save them
saveRDS(evl.markers.bystage, "cascades/aucpr/EVL/Periderm.rds")
saveRDS(pgc.markers.bystage, "cascades/aucpr/Primordial Germ Cells.rds")

# Separate actual marker lists from the stats lists
gene.markers.de <- lapply(gene.markers, function(x) x[[1]])
gene.markers.stats <- lapply(gene.markers[1:23], function(x) x[[2]])
names(gene.markers.de) <- names(gene.markers)
names(gene.markers.stats) <- names(gene.markers)[1:23]

# Add them PGC and EVL to gene markers DE
gene.markers.de[["Primordial Germ Cells"]] <- pgc.markers.bystage$diff.exp
gene.markers.de[["EVL/Periderm"]] <- evl.markers.bystage$diff.exp
```

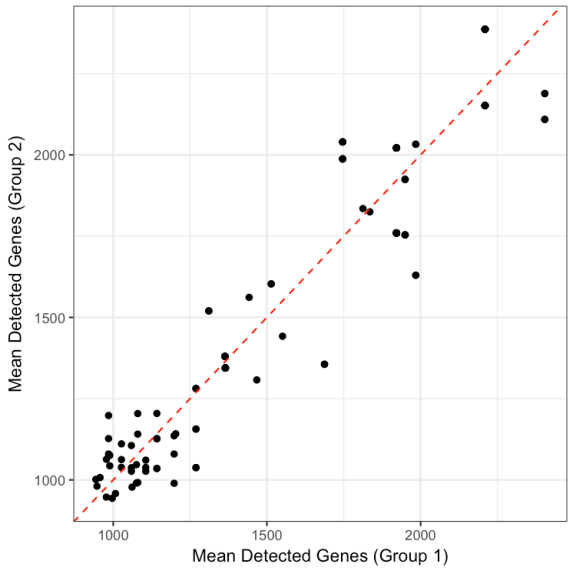
Differential expression should not be biased by library complexity

Since the size of our cells varies with developmental stage, so does the RNA content, and generally the number of recovered transcripts and detected genes. Since genes could be detected as differentially expressed due to technical effects where they were detected in higher complexity transcriptomes, but dropped out of more sparse transcriptomes, we wanted to make sure this was not a problem with our differential expression testing. Thus, we tracked the average number of transcripts and genes per cell in each of the differential expression tests performed during construction of the gene expression cascades. We find that, because cell populations are matched in pseudotime prior to calculating differential expression, they are also largely matched in number of transcripts and genes detected, so we do not expect this to pose a problem.

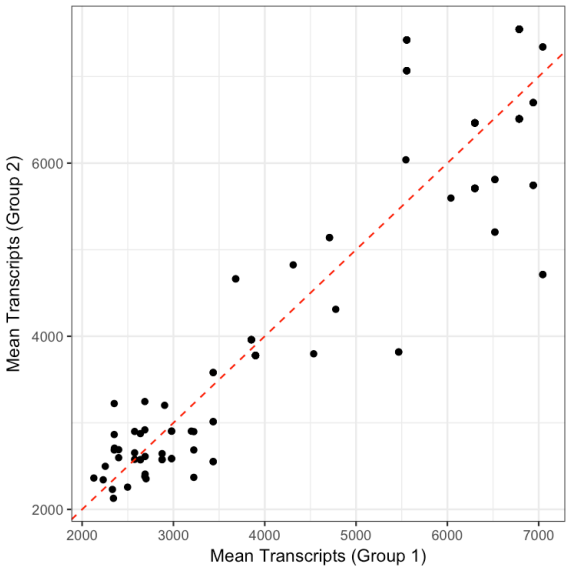
```
# Compile all comparison stats into a single table
all.de.stats <- do.call("rbind", gene.markers.stats)

# Do a few plots
ggplot(all.de.stats, aes(x = pt.1.mean, y = pt.2.mean)) + geom_point() + theme_bw() +
    geom_abline(slope = 1, intercept = 0, col = "red", lty = 2) + labs(x = "Mean Pseudotime (Group 1)",
    y = "Mean Pseudotime (Group 2)")
```




```
ggplot(all.de.stats, aes(x = genes.1.mean, y = genes.2.mean)) + geom_point() + theme_bw() +
    geom_abline(slope = 1, intercept = 0, col = "red", lty = 2) + labs(x = "Mean Detected Genes (Group 1)",
    y = "Mean Detected Genes (Group 2)")
```



```
ggplot(all.de.stats, aes(x = trans.1.mean, y = trans.2.mean)) + geom_point() + theme_bw() +
    geom_abline(slope = 1, intercept = 0, col = "red", lty = 2) + labs(x = "Mean Transcripts (Group 1)",
    y = "Mean Transcripts (Group 2)")
```



NMF module comparison along tree

We also identified chains of connected NMF modules that were upregulated at branchpoints in the data, and considered the top 25 genes loaded in the module to also be part of the gene cascade for that trajectory.

Load the NMF data

```
# Load the data
cm <- read.csv("~/Dropbox/Jeff-Yiqun/DE modules/AllModuleByAllCell.csv", row.names = 1)

# Load top genes for each module
what.loaded <- load("~/Dropbox/Jeff-Yiqun/DE modules/Module_top_25genes.Robj")
mod.genes.top25 <- top_25genes
what.loaded <- load("~/Dropbox/Jeff-Yiqun/DE modules/module_lineages.Robj")
ml <- all_lineages
rm(list = c("what.loaded", "all_lineages", "top_25genes"))
```

We evaluated each segment of unbranched modules in the connected module tree.

```
# Create a list of segments of connected modules without branches.
mod.lin.segs <- list(ZF3S_22 = ml[["3S_22"]][1:5], ZF6S_29 = "6S_29", ZF6S_9 = "6S_9",
    ZF3S_23 = c("3S_23", "B_16"), ZFB_24 = "B_24", ZF6S_35 = ml[["6S_35"]][1:3],
    ZF6S_14 = c("6S_14", "3S_13"), ZF6S_16 = c("6S_16", "3S_14"), ZFB_13 = c("B_13",
        "90_16"), ZF90_26 = "90_26", ZF90_8 = "90_8", ZF75_9 = c("75_9", "60_20"),
    ZFS_5 = c("S_5"), ZF50_11 = "50_11", ZF6S_15 = ml[["6S_15"]][1:4], ZF6S_13 = "6S_13",
    ZF6S_34 = "6S_34", ZF3S_12 = c("3S_12", "B_9", "90_13"), ZF75_14 = "75_14", ZF6S_23 = ml[["6S_23"]][1:5],
    ZF60_16 = c("60_16", "S_14"), ZF90_5 = "90_5", ZF6S_10 = ml[["6S_10"]][1:8],
    ZF6S_1 = "6S_1", ZF6S_27 = "6S_27", ZF3S_1 = ml[["6S_1"]][2:8], ZF6S_40 = ml[["6S_40"]][1:3],
    ZF6S_20 = ml[["6S_20"]][1:3], ZF90_27 = ml[["90_27"]][1:3], ZF90_25 = ml[["6S_20"]][4:6],
    ZFS_3 = c("S_3", "50_6"), ZF6S_3 = ml[["6S_3"]][1:8], ZF6S_2 = ml[["6S_2"]][1:3],
    ZF6S_17 = ml[["6S_17"]][1:3], ZF90_1 = c("90_1", "75_1"), ZF6S_18 = "6S_18",
    ZF6S_5 = "6S_5", ZF3S_15 = ml[["6S_5"]][2:5], ZF60_1 = "60_1", ZF6S_22 = c("6S_22",
        "3S_19"), ZF6S_7 = c("6S_7", "3S_9"), ZFB_14 = c("B_14", "90_20"), ZF6S_21 = ml[["6S_21"]][1:4],
    ZF75_11 = c("75_11", "60_18"), ZFS_1 = "S_1", ZF50_2 = "50_2", ZF6S_26 = ml[["6S_26"]][1:7],
    ZF6S_4 = ml[["6S_4"]][1:8], ZF75_22 = ml[["75_22"]][1:4], ZF90_28 = ml[["90_28"]][1:5])
```

And then used t-tests along the structure of the URD dendorgram to find modules that were enriched in particular trajectories.

```
# Create a module matrix that only includes those modules that are in the
# segments you just defined.
cm.goodtree <- cm[unlist(mod.lin.segs), ]
```

```
# Do the tests for everything except the EVL & PGCs, which need a different root
# parameter.
tips.to.run <- as.character(object@tree$segment.names)
root.to.use <- c(rep("81", 23), "82", NA)  # Use different root for PGC & EVL, because they're hooked up outside the blastoderm t
nmf.markers <- list()
for (tipn in 1:length(tips.to.run)) {
    tip <- tips.to.run[tipn]
    root <- root.to.use[tipn]
    if (is.na(root))
        root <- NULL
    print(paste0(Sys.time(), ": Starting ", tip))
    markers <- moduleTestAlongTree(object, tips = tip, data = cm.goodtree, genelist = mod.genes.top25,
        pseudotime = "pseudotime", exclude.upstream = T, effect.size = log(4), p.thresh = 0.01,
        root = root, min.expression = 0.05)
    saveRDS(markers, file = paste0("cascades/nmf/", tip, ".rds"))
    nmf.markers[[tip]] <- markers
}
```

```
## [1] "2018-02-24 21:15:15: Starting Spinal Cord"
## [1] "2018-02-24 21:16:02: Starting Diencephalon"
## [1] "2018-02-24 21:16:50: Starting Optic Cup"
## [1] "2018-02-24 21:17:36: Starting Midbrain+Neural Crest"
## [1] "2018-02-24 21:18:22: Starting Hindbrain R3"
## [1] "2018-02-24 21:19:05: Starting Hindbrain R4+5+6"
## [1] "2018-02-24 21:19:47: Starting Telencephalon"
## [1] "2018-02-24 21:20:31: Starting Epidermis"
## [1] "2018-02-24 21:21:15: Starting Neural Plate Border"
## [1] "2018-02-24 21:22:00: Starting Placode Adeno.+Lens+Trigeminal"
## [1] "2018-02-24 21:22:42: Starting Placode Epibranchial+Otic"
## [1] "2018-02-24 21:23:25: Starting Placode Olfactory"
## [1] "2018-02-24 21:24:07: Starting Tailbud"
## [1] "2018-02-24 21:24:34: Starting Adaxial Cells"
## [1] "2018-02-24 21:24:59: Starting Somites"
## [1] "2018-02-24 21:25:23: Starting Hematopoeitic (ICM)"
## [1] "2018-02-24 21:25:45: Starting Hematopoeitic (RBI)+Pronephros"
## [1] "2018-02-24 21:26:08: Starting Endoderm Pharyngeal"
## [1] "2018-02-24 21:26:30: Starting Endoderm Pancreatic+Intestinal"
## [1] "2018-02-24 21:26:52: Starting Heart Primordium"
## [1] "2018-02-24 21:27:12: Starting Cephalic Mesoderm"
## [1] "2018-02-24 21:27:32: Starting Prechordal Plate"
## [1] "2018-02-24 21:27:43: Starting Notochord"
## [1] "2018-02-24 21:27:54: Starting Primordial Germ Cells"
## [1] "2018-02-24 21:28:23: Starting EVL/Periderm"
```

Combine the two sets of markers

We combined genes identified by URD's differential expression testing and by NMF module loading into a single set of markers for each trajectory.

```
### Combine the two sets of markers
tips.to.run <- as.character(object@tree$segment.names)
combined.gene.markers <- lapply(tips.to.run, function(tip) {
    gm <- rownames(gene.markers.de[[tip]])
    nm <- nmf.markers[[tip]]$genes
    return(unique(c(gm, nm)))
})
names(combined.gene.markers) <- tips.to.run
```

Impulse fits

We then fit the expression of each marker gene in each trajectory using an impulse model to determine the timing of onset and offset of its expression to order genes in the cascade. Cells in each trajectory are grouped using a moving window through pseudotime; then, mean gene expression is then calculated in each window, and scaled to the maximum mean expression observed in the trajectory. Then, a linear model, single onset sigmoid model, and convex and concave double sigmoid models are fit to the data, and the best fit is chosen by minimizing the sum of squared residuals, penalized according to the complexity of the model. The parameters of the chosen model (for instance, the inflection point of a sigmoid) are then used

to calculate genes' onset time, which is then used to order genes. Importantly, the double sigmoid models allow for accurate fitting of genes that are expressed in a brief pulse (a convex double sigmoid, or "impulse"), or that are maternally loaded, decrease over time, and then are re-expressed in particular trajectories (a concave double sigmoid).

```
# Generate impulse fits
gene.cascades.combined <- lapply(tips.to.run, function(tip) {
    print(paste0(Sys.time(), ": Impulse Fit ", tip))
    seg.cells <- cellsAlongLineage(object, tip, remove.root = F)
    casc <- geneCascadeProcess(object = object, pseudotime = "pseudotime", cells = seg.cells,
        genes = combined.gene.markers[[tip]], moving.window = 5, cells.per.window = 18,
        limit.single.sigmoid.slopes = "on", verbose = F)
    tip.file.name <- gsub("/", "_", tip)
    saveRDS(casc, file = paste0("cascades/impulse/casc_", tip.file.name, ".rds"))
    return(casc)
})
names(gene.cascades.combined) <- tips.to.run
```

```
## [1] "2018-02-24 21:28:51: Impulse Fit Spinal Cord"
## [1] "2018-02-24 21:37:46: Impulse Fit Diencephalon"
## [1] "2018-02-24 21:42:12: Impulse Fit Optic Cup"
## [1] "2018-02-24 21:45:53: Impulse Fit Midbrain+Neural Crest"
## [1] "2018-02-24 21:50:03: Impulse Fit Hindbrain R3"
## [1] "2018-02-24 21:55:00: Impulse Fit Hindbrain R4+5+6"
## [1] "2018-02-24 21:58:10: Impulse Fit Telencephalon"
## [1] "2018-02-24 22:03:20: Impulse Fit Epidermis"
## [1] "2018-02-24 22:13:14: Impulse Fit Neural Plate Border"
## [1] "2018-02-24 22:26:43: Impulse Fit Placode Adeno.+Lens+Trigeminal"
## [1] "2018-02-24 22:34:39: Impulse Fit Placode Epibranchial+Otic"
## [1] "2018-02-24 22:40:56: Impulse Fit Placode Olfactory"
## [1] "2018-02-24 22:51:02: Impulse Fit Tailbud"
## [1] "2018-02-24 22:55:21: Impulse Fit Adaxial Cells"
## [1] "2018-02-24 22:59:04: Impulse Fit Somites"
## [1] "2018-02-24 23:01:32: Impulse Fit Hematopoeitic (ICM)"
## [1] "2018-02-24 23:04:38: Impulse Fit Hematopoeitic (RBI)+Pronephros"
## [1] "2018-02-24 23:06:01: Impulse Fit Endoderm Pharyngeal"
## [1] "2018-02-24 23:08:03: Impulse Fit Endoderm Pancreatic+Intestinal"
## [1] "2018-02-24 23:10:45: Impulse Fit Heart Primordium"
## [1] "2018-02-24 23:12:11: Impulse Fit Cephalic Mesoderm"
## [1] "2018-02-24 23:14:07: Impulse Fit Prechordal Plate"
## [1] "2018-02-24 23:16:20: Impulse Fit Notochord"
## [1] "2018-02-24 23:18:13: Impulse Fit Primordial Germ Cells"
## [1] "2018-02-24 23:18:34: Impulse Fit EVL/Periderm"
```

Heatmaps

We then plotted each trajectory's gene cascade in a heatmap. Genes are ordered along the y-axis, according to our determined time of expression onset. Along the x-axis is the progression of pseudotime. Plotted is the scaled mean expression within each pseudotime moving window. We determined which genes were recovered from differential expression testing by URD, or were members of a connected NMF gene module that was upregulated in a particular trajectory. This is plotted next to the heatmap, colored red for genes exclusively identified by NMF, blue for genes exclusively identified by URD, and purple for genes identified by both approaches. (These plots were output to a PDF, but we show an example of one below.)

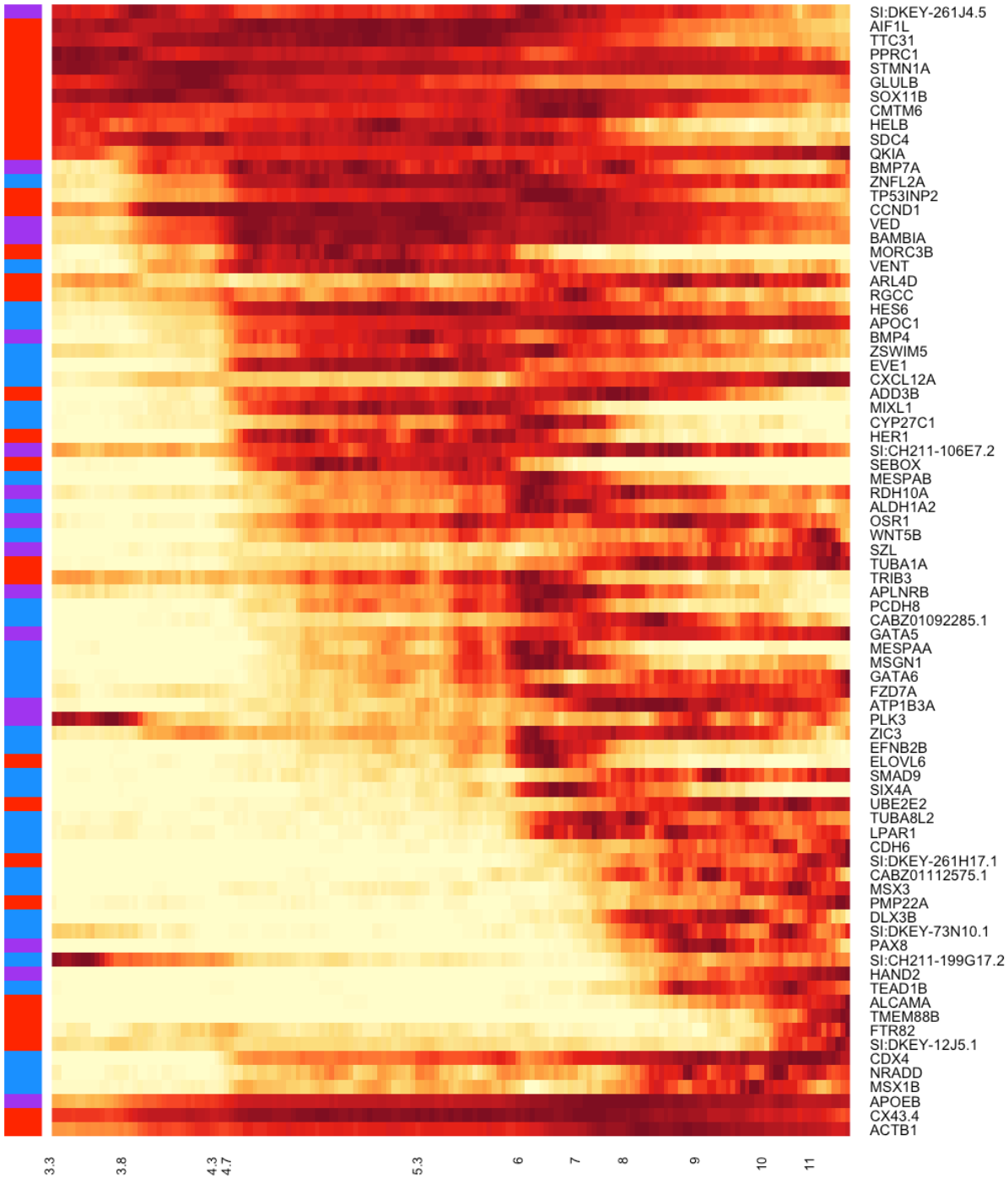
```
# Generate color bars for NMF vs. URD found markers
urd.nmf.markers <- lapply(tips.to.run, function(tip) {
    gm <- rownames(gene.markers.de[[tip]])
    nm <- nmf.markers[[tip]]$genes
    return(list(red = setdiff(nm, gm), purple = intersect(nm, gm), dodgerblue1 = setdiff(gm,
        nm)))
})
names(urd.nmf.markers) <- tips.to.run

# Make a heatmap of every cascade in a single PDF.
pdf(file = "cascades/cascades.pdf", width = 7.5, height = 10)
for (tip in tips.to.run) {
    geneCascadeHeatmap(cascade = gene.cascades.combined[[tip]], color.scale = RColorBrewer::brewer.pal(9,
        "YlOrRd"), add.time = "HPF", times.annotate = c(3.3, 3.8, 4.3, 4.7, 5.3,
        6, 7, 8, 9, 10, 11, 12), title = tip, annotation.list = urd.nmf.markers[[tip]])
}
```

```
dev.off()
```

```
## quartz_off_screen
##                 2
```

```
tip <- "Heart Primordium"
geneCascadeHeatmap(cascade = gene.cascades.combined[[tip]], color.scale = RColorBrewer::brewer.pal(9,
    "YlOrRd"), add.time = "HPF", times.annotate = c(3.3, 3.8, 4.3, 4.7, 5.3, 6, 7,
    8, 9, 10, 11, 12), title = tip, annotation.list = urd.nmf.markers[[tip]])
```

Heart Primordium



URD 7: Plotting

```
library(URD)
library(rgl)
library(gridExtra)  # For arranging plots
library(RColorBrewer)  # For color palettes

# Set up knitr to capture rgl output
rgl::setupKnitr()
```

Load previous saved object

```
object <- readRDS("obj/object_6_tree.rds")
```

Color Palettes

Define some color palettes to use for figures. Most of these are gentle modifications of RColorBrewer palettes.

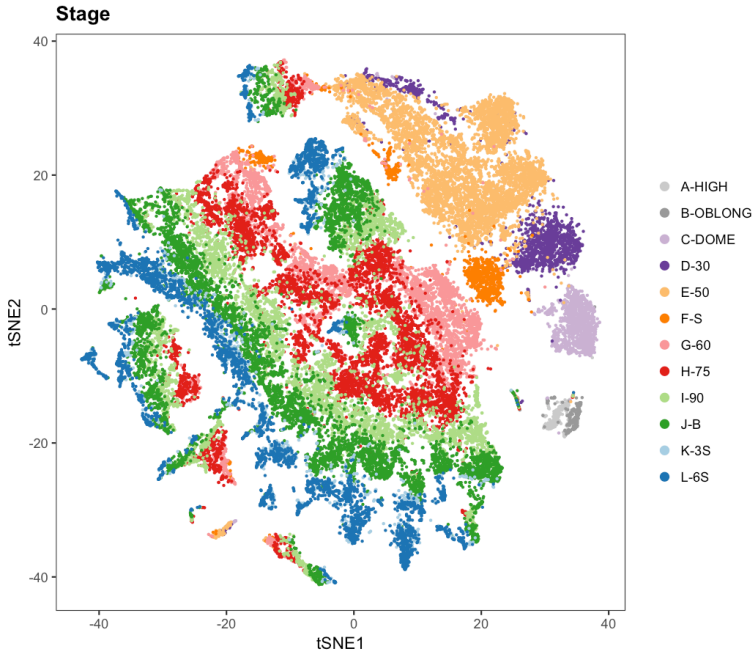
```
# Colors to use for stage
stage.colors <- c("#CCCCCC", RColorBrewer::brewer.pal(9, "Set1")[9], RColorBrewer::brewer.pal(12,
    "Paired")[c(9, 10, 7, 8, 5, 6, 3, 4, 1, 2)])

# Preference colors for the preference plots
pref.colors <- c("#CECECE", "#CBDAC2", RColorBrewer::brewer.pal(9, "YlGnBu")[3:9])

# Red-orange color scheme for gene expression
fire.with.grey <- c("#CECECE", "#DDC998", RColorBrewer::brewer.pal(9, "YlOrRd")[3:9])

# Grey-blue-green color scheme for module and gene expression
pond.with.grey <- c("#CECECE", "#CBDAC2", RColorBrewer::brewer.pal(9, "YlGnBu")[3:9])

branch.colors <- c("#CECECE", "#E6298B")
```

plotDim

The **plotDim** command allows plotting cells according to various dimensionality reductions that have been performed (such as tSNE, PCA, or the diffusion map.)

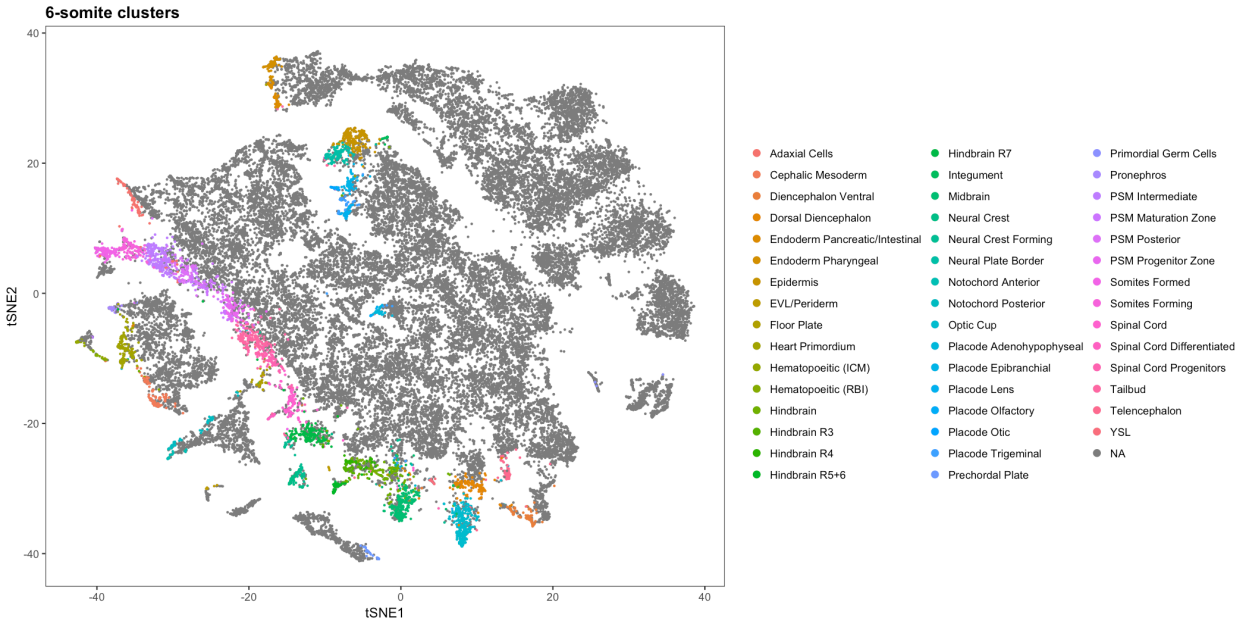
tSNE - Stage

```
plotDim(object, "stage.nice", reduction.use = "tSNE", discrete.colors = stage.colors,
    plot.title = "Stage")
```



tSNE - Clustering

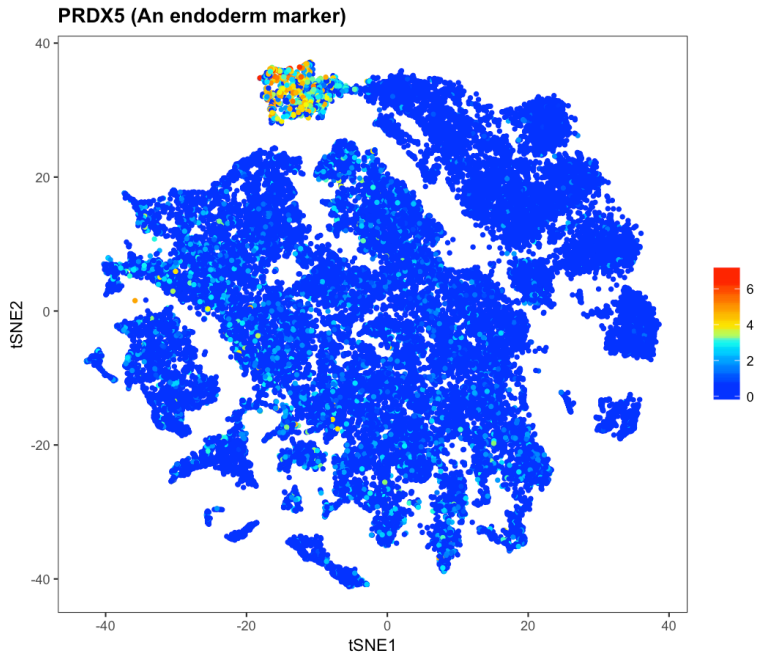
```
plotDim(object, "ZF6S-Cluster", reduction.use = "tSNE", plot.title = "6-somite clusters")
```



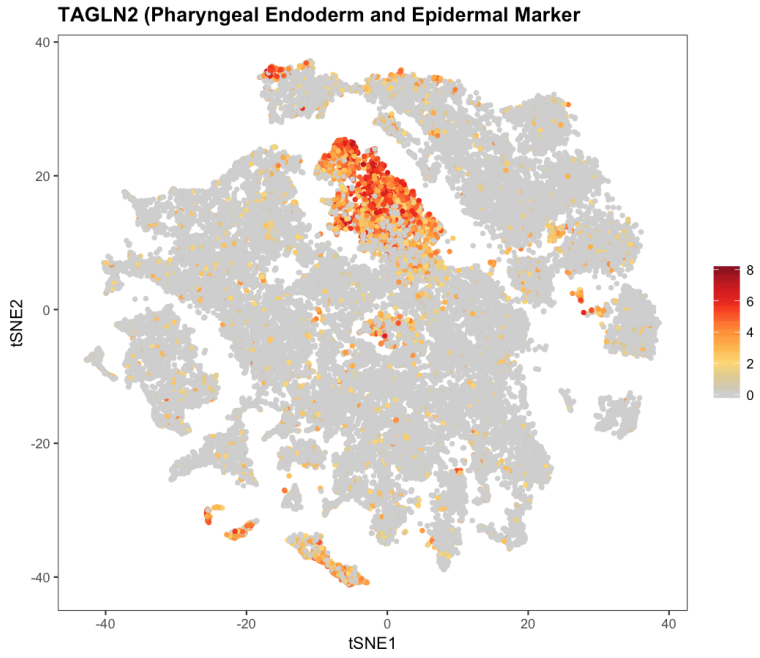
tSNE - Gene Expression

Individual markers can be plotted with the default blue-to-red color scheme, or a custom palette.

```
plotDim(object, "PRDX5", reduction.use = "tSNE", plot.title = "PRDX5 (An endoderm marker)")
```

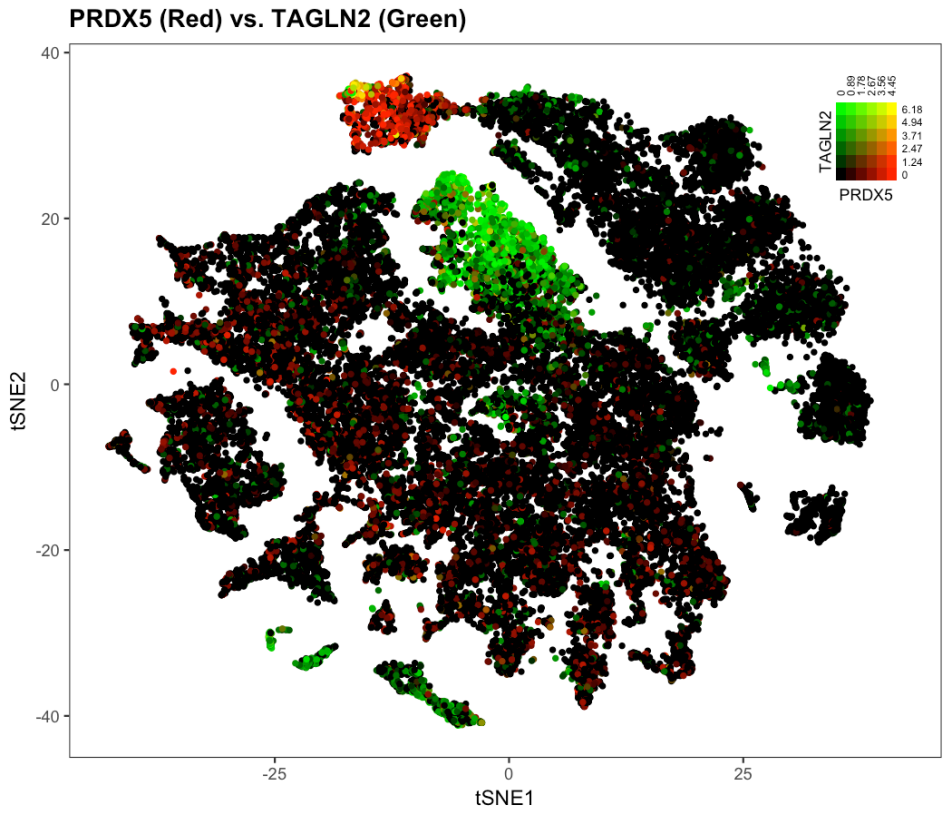



```
plotDim(object, "TAGLN2", reduction.use = "tSNE", plot.title = "TAGLN2 (Pharyngeal Endoderm and Epidermal Marker",
    colors = fire.with.grey)
```



Two markers can be simultaneously plotted using a red-green color scheme.

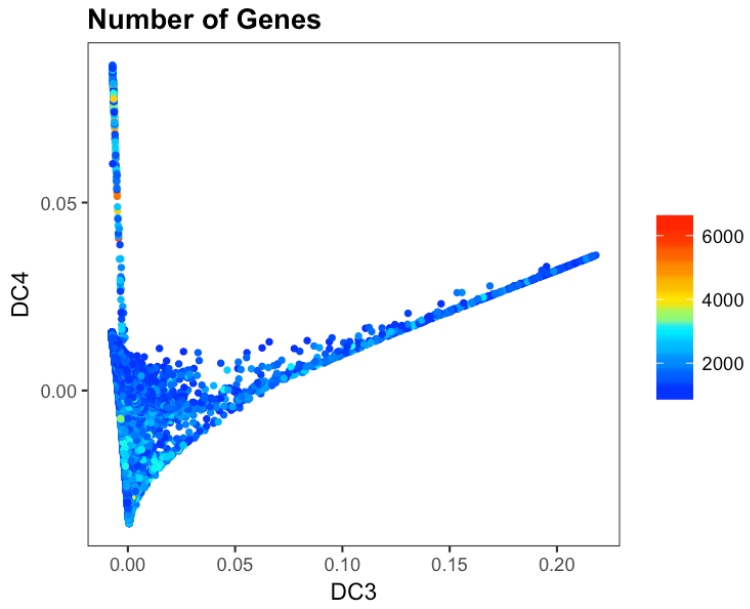
```
plotDimDual(object, label.red = "PRDX5", label.green = "TAGLN2", reduction.use = "tSNE",
    plot.title = "PRDX5 (Red) vs. TAGLN2 (Green)", legend.offset.x = 5)
```



Diffusion Map

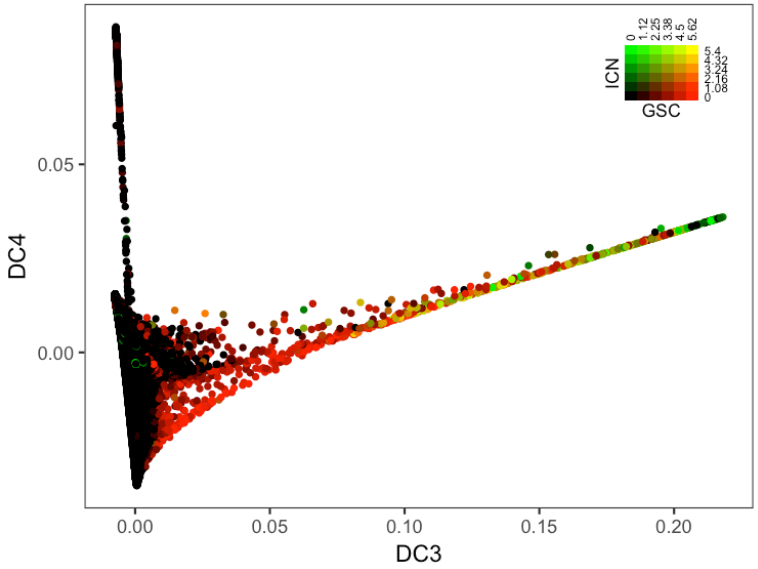
Pairs of components from the diffusion map (or PCA, if reduction.use="pca") can also be plotted with any metadata or gene expression.

```
plotDim(object, reduction.use = "dm", dim.x = 3, dim.y = 4, label = "n.Genes", plot.title = "Number of Genes")
```



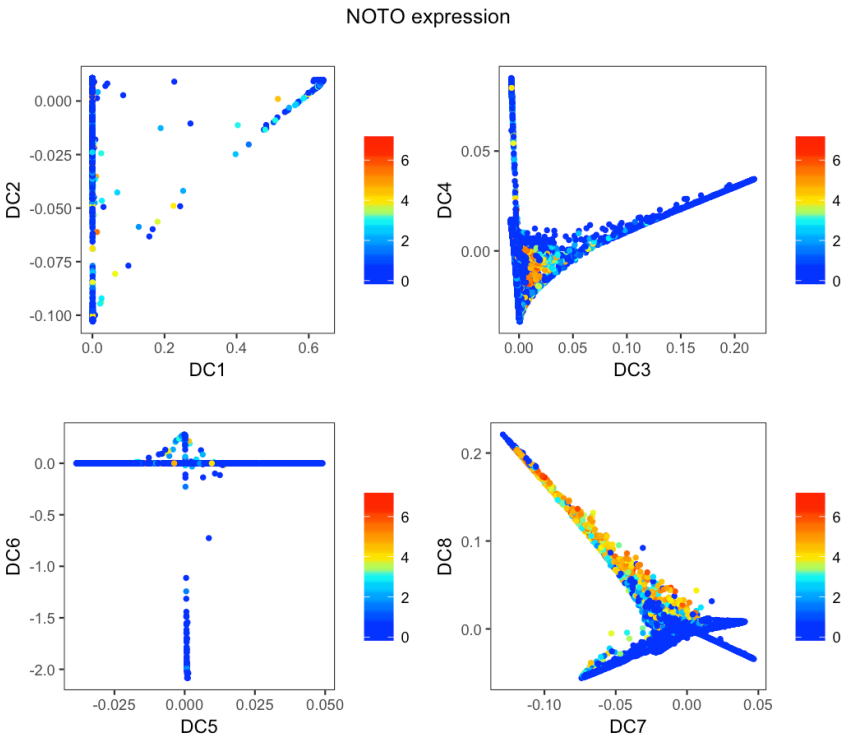
Additionally, pairs of labels can be plotted in a red-green color scheme.

```
plotDimDual(object, reduction.use = "dm", dim.x = 3, dim.y = 4, label.red = "GSC",
    label.green = "ICN")
```



Furthermore, any plotDim command can be run against many pairs of components using plotDimArray – all parameters are passed to plotDim, and the x and y dimensions are taken in pairs from the dims.to.plot vector.

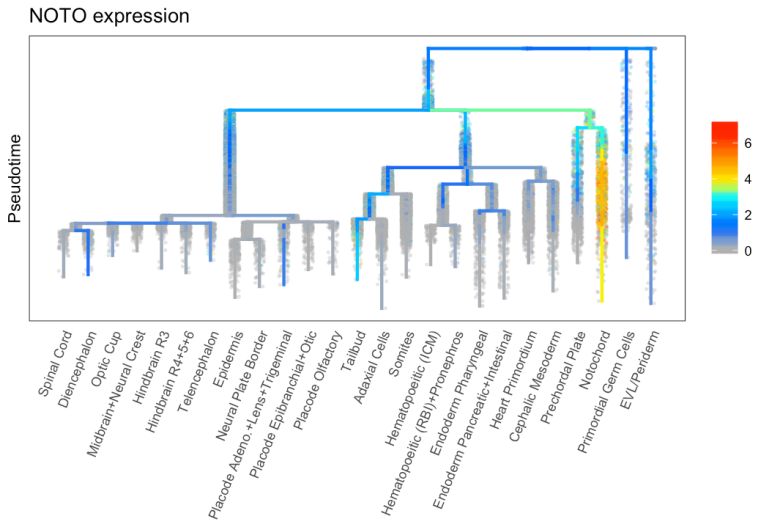
```
plotDimArray(object, reduction.use = "dm", dims.to.plot = 1:8, label = "NOTO", outer.title = "NOTO expression",
    plot.title = "")
```



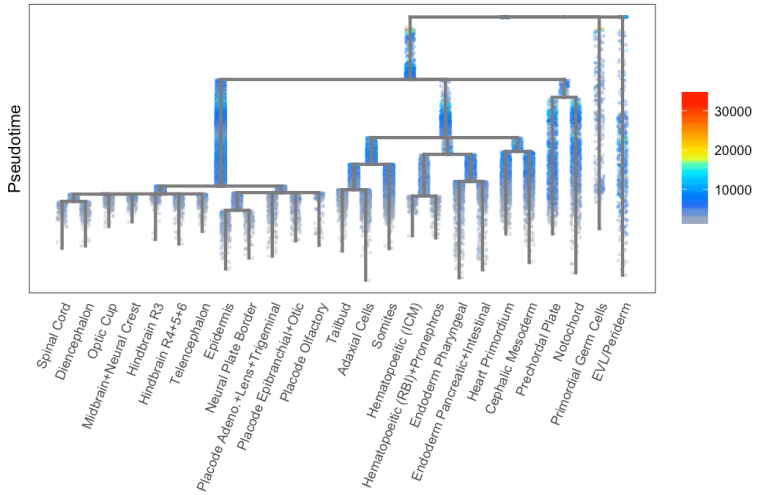
Tree Dendrogram

URD's recovered tree dendrogram can also be decorated with any gene expression or metadata.

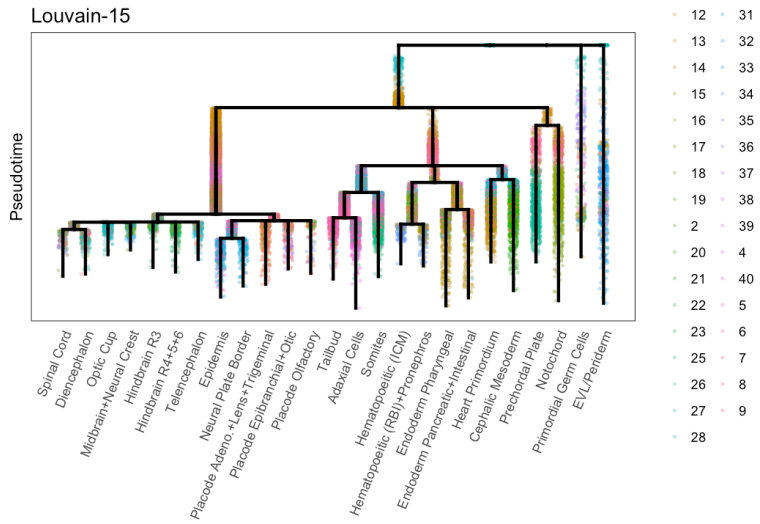
```
plotTree(object, "NOTO", title = "NOTO expression")
```




```
plotTree(object, "n.Trans", title = "")
```



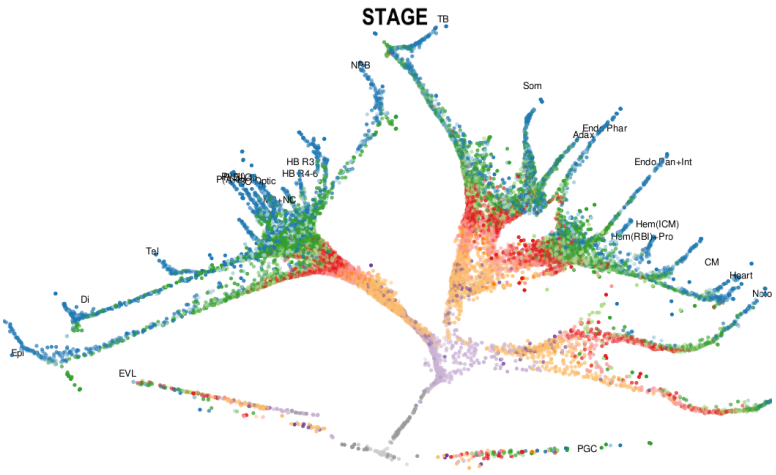

```
plotTree(object, "Louvain-15")
```



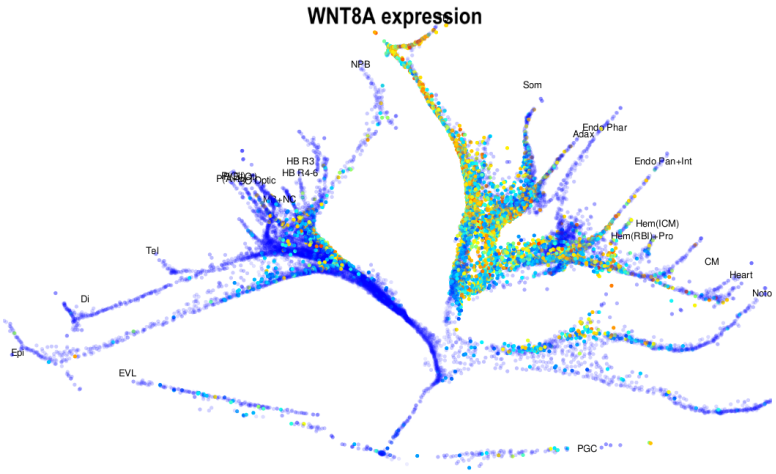
Force-directed Layout

Similarly, the force-directed layout can be decorated with any clustering, gene expression, or metadata.

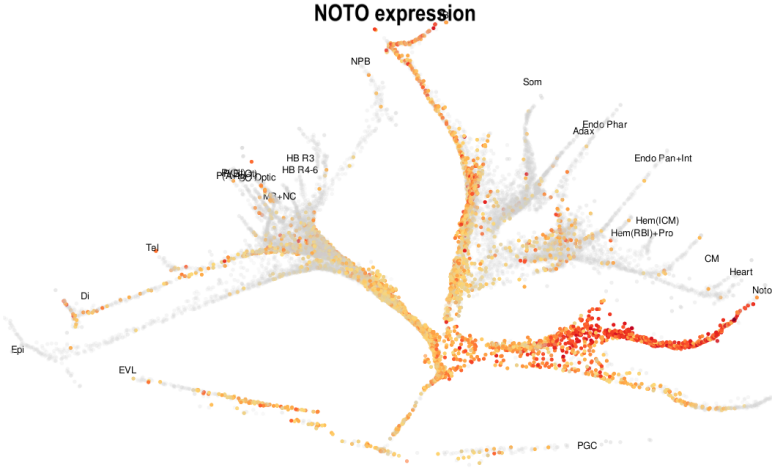
```
plotTreeForce(object, "stage.nice", title = "STAGE", title.line = 1, discrete.colors = stage.colors,
    alpha = 0.4)
```



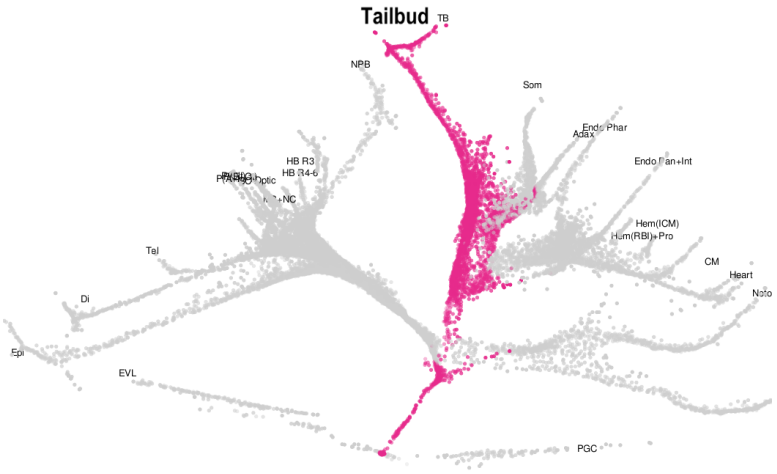

```
plotTreeForce(object, "WNT8A", title = "WNT8A expression", title.line = 1)
```




```
plotTreeForce(object, "NOTO", title = "NOTO expression", title.line = 1, colors = fire.with.grey)
```




```
object <- groupFromCells(object, group.id = "lineage_Tailbud", cells = cellsAlongLineage(object,
    "Tailbud", remove.root = F))
plotTreeForce(object, "lineage_Tailbud", title = "Tailbud", title.line = 1, discrete.colors = branch.colors,
    alpha = 0.4)
```

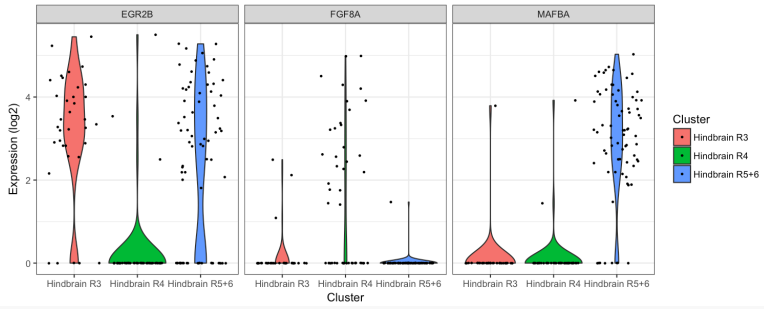


Gene Expression

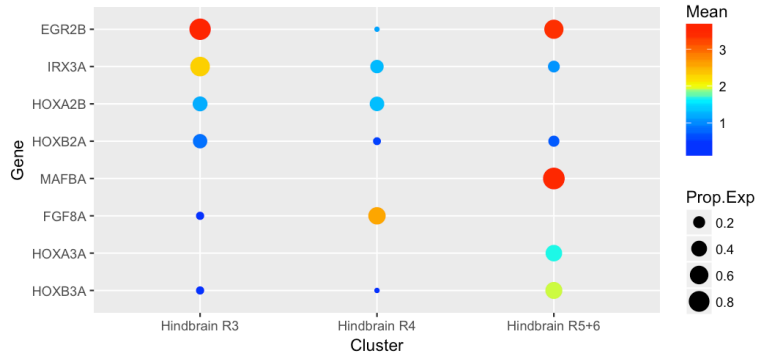
We also include several functions for exploring gene expression within lineages or populations. Here we use a violin plot and dot plot to illustrate markers of the clusters used as hindbrain tips.

```
plotViolin(object, labels.plot = c("EGR2B", "FGF8A", "MAFBA"), clustering = "ZF6S-Cluster",
    clusters = c("Hindbrain R3", "Hindbrain R4", "Hindbrain R5+6"))
```

```
## Warning in plotViolin(object, labels.plot = c("EGR2B", "FGF8A", "MAFBA"), :
## NAs introduced by coercion
```




```
plotDot(object, genes = c("EGR2B", "IRX3A", "HOXA2B", "HOXB2A", "MAFBA", "FGF8A",
    "HOXA3A", "HOXB3A"), clustering = "ZF6S-Cluster", clusters.use = c("Hindbrain R3",
    "Hindbrain R4", "Hindbrain R5+6"))
```



Markers of a specific lineage

In the manuscript, we illustrated the expression of the determined markers for a single cascade on the force-directed layout (**Figure S5** shows markers of the prechordal plate). We took all genes that were part of the prechordal plate gene cascade and applied an additional layer of specificity – we required that they were ~4 times better than a random precision-recall classifier when compared globally between the prechordal plate cascade and the rest of the embryo. We then determined whether they had already been annotated as having expression in the prechordal plate (or one of its synonyms) in ZFIN (the Zebrafish Information Network). Finally, we plotted the expression of each gene, colored according to whether it had been previously annotated or not.

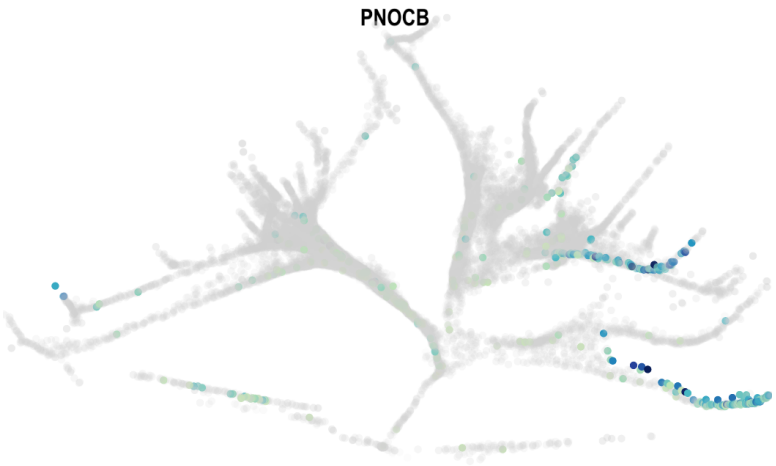
```
# Load gene cascade
pcp.cascade <- readRDS("cascades/impulse/casc_Prechordal Plate.rds")
pcp.markers <- rownames(pcp.cascade$scaled.expression)

# Determine which genes are also global markers
pcp.axial.cells <- cellsInCluster(object, "segment", c("29", "79"))
pcp.markers.global <- markersAUCPR(object, cells.1 = pcp.axial.cells, genes.use = pcp.markers)
marker.thresh <- aucprThreshold(cells.1 = pcp.axial.cells, cells.2 = setdiff(unlist(object@tree$cells.in.segment),
    pcp.axial.cells), factor = 2.5, max.auc = Inf)
pcp.de.markers <- pcp.markers.global[pcp.markers.global$AUCPR >= marker.thresh, ]

# Who is annotated already in ZFIN?  Search terms: anterior axial hypoblast,
# prechordal plate, polster, hatching gland
zfin.pcp.markers <- read.csv(file = "data/ZFIN_annotated_Markers.csv", header = F)
new.markers <- setdiff(rownames(pcp.de.markers), toupper(zfin.pcp.markers$V1))

# Still had to hand verify these in ZFIN, since sometimes genes have been renamed
# or have a weird but equivalent anatomy term (like 'dorsal marginal
# blastomeres')
renamed.in.zfin <- c("HE1A", "HE1B", "LGALS3L", "SHISA2", "CHD", "OTX1A", "OTX1B",
    "ATPIF1B", "FBXO2")
known.marker <- c("RIPPLY1", "NDR1", "LHX1A", "MIXL1", "ISM1", "FSCN1A", "CTH1",
    "DHRS3B", "SND1")  # Genes with semi-random, but equivalent anatomy terms

# Final new/old markers list.
new.markers <- setdiff(new.markers, c(renamed.in.zfin, known.marker))  # 49
old.markers <- setdiff(rownames(pcp.de.markers), new.markers)  # 71
pcp.markers <- c(new.markers, old.markers)  # 120

# Order markers according to gene cascade
timing <- pcp.cascade$timing[pcp.markers, ]
timing[intersect(which(is.na(timing$time.on)), which(is.infinite(timing$time.off))),
    "time.on"] <- Inf
ordered.markers <- pcp.markers[order(timing$time.on, timing$time.off, na.last = F)]
ordered.markers.new <- ordered.markers %in% new.markers

# Save the list of markers for later.
write(ordered.markers, "cascades/pcp_markers_FigS5.txt")
```

We plot all of the markers to a folder for building the supplemental figure…

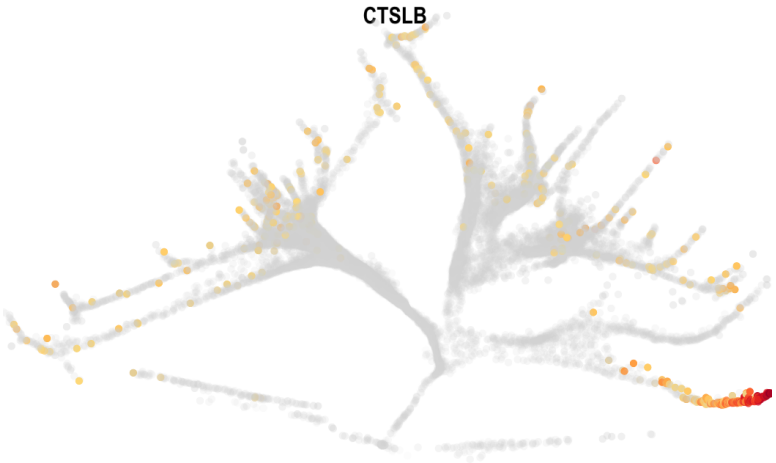
```
# Plot each marker, colored based on whether it was annotated in ZFIN previously.
for (i in 1:length(ordered.markers)) {
    m <- ordered.markers[i]
    if (ordered.markers.new[i])
        colors.use <- pond.with.grey else colors.use <- fire.with.grey
    plotTreeForce(object, m, alpha = 0.7, alpha.fade = 0.08, size = 10, density.alpha = T,
        label.tips = F, colors = colors.use, view = "figure1")
    text3d(x = -8, y = 4.2, z = 100, m, cex = 4)
    Sys.sleep(0.2)
    rgl.snapshot(file = paste0("cascades/pcp_markers/", sprintf("%03d", i), "-",
        m, ".png"))
    rgl.close()
}
```

… but include here a couple of examples of a new marker (*pnocb*) and a very classic marker of the prechordal plate (*ctslb/hgg1*):

```
plotTreeForce(object, ordered.markers[71], title = ordered.markers[71], alpha = 0.7,
    alpha.fade = 0.08, size = 10, density.alpha = T, label.tips = F, colors = pond.with.grey,
    view = "figure1")
```




```
plotTreeForce(object, label = ordered.markers[72], title = ordered.markers[72], alpha = 0.7,
    alpha.fade = 0.08, size = 10, density.alpha = T, label.tips = F, colors = fire.with.grey,
    view = "figure1")
```



Preference plot at a branchpoint

In the manuscript, we described that the axial mesoderm branchpoint (between the notochord and prechordal plate) has cells that expressed genes characteristic of both downstream populations (**Figure 6**). We illustrated this using preference plots, colored by gene expression.

Define the preference layout for the branchpoint

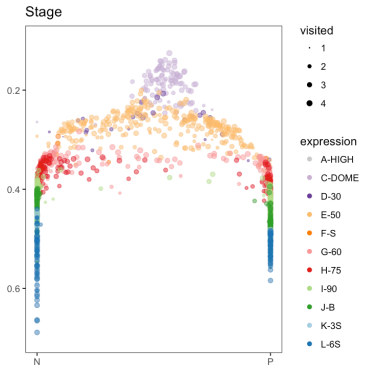
First, the preference plot layout is defined. Cells that were visited by random walks started in the two tips in question are included in the layout. Cells are ordered along the y-axis according to pseudotime. Cells are placed along the x-axis based on their preference, which is based on their ratio of visits by the random walks from each tip.

```
# Define layout for plots
np.layout <- branchpointPreferenceLayout(object, pseudotime = "pseudotime", lineages.1 = "29",
    lineages.2 = "32", parent.of.lineages = "79", opposite.parent = c("72", "78"),
    min.visit = 1)
```

Plot stage on the preference plot

We found that the intermediate cells were prevalent at mid-gastrulation, from 60%-90% epiboly.

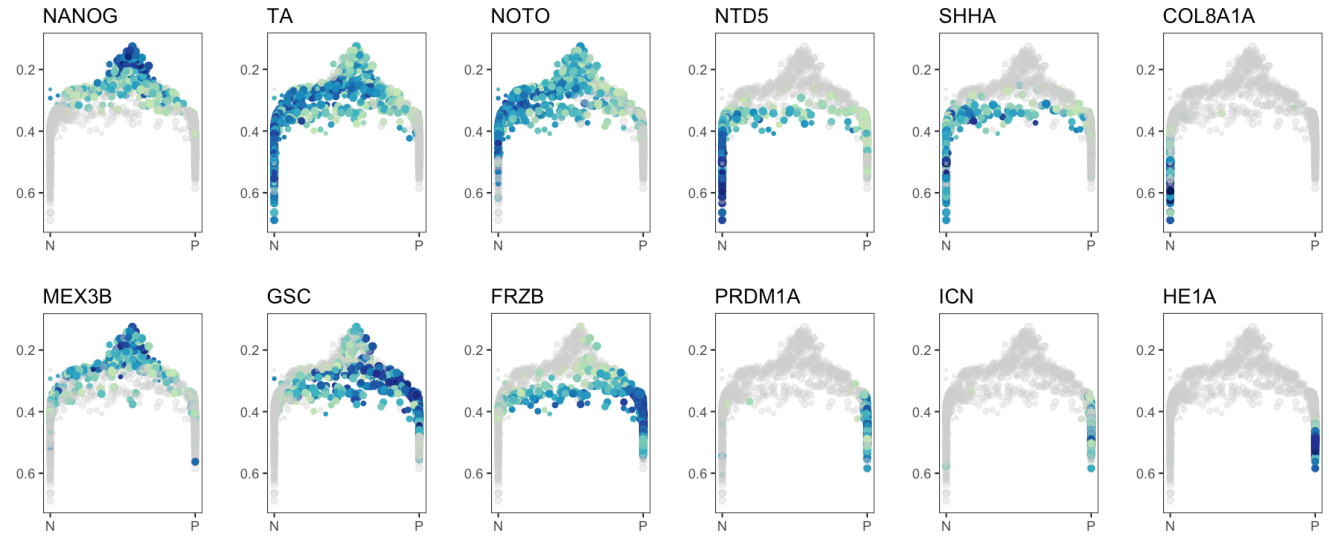
```
plotBranchpoint(object, np.layout, label = "stage.nice", point.alpha = 0.5, populations = c("P",
    "N"), pt.lim = c(0.7, 0.1), xlab = "", ylab = "", legend = T, axis.lines = F,
    fade.low = 0, discrete.colors = stage.colors[c(1, 3:12)], title = "Stage")
```



Plot gene expression on the branchpoint.

We found that the intermediate cells no longer expressed progenitor markers (*nanog*, *mex3b*), expressed early markers of both cell types (*ta*, *noto*, *gsc*, and *frzb*), and expressed late markers of the notochord (*ntd5* and *shha*), but did not express late markers of the prechordal plate (*prdm1a*, *icn*). By the time that differentiation genes (*col8a1a* and *he1a*) were expressed, there were no longer intermediate cells between the two trajectories.

```
# Define genes to plot
axial.genes.plot <- c("NANOG", "TA", "NOTO", "NTD5", "SHHA", "COL8A1A", "MEX3B",
    "GSC", "FRZB", "PRDM1A", "ICN", "HE1A")

# Plot gene expression on the branchpoint preference plot
axial.branchpoint.plots <- lapply(axial.genes.plot, function(gene) plotBranchpoint(object,
    np.layout, label = gene, point.alpha = 1, populations = c("P", "N"), pt.lim = c(0.7,
        0.11), color.scale = pref.colors, xlab = "", ylab = "", title = gene, legend = F,
    axis.lines = F, fade.low = 0.66))

grid.arrange(grobs = axial.branchpoint.plots, ncol = 6)
```



URD: Choosing Parameters - Diffusion Map Sigma

```
library(URD)
```

Load previous saved object

```
object <- readRDS("obj/object_2_trimmed.rds")
```

Calculate diffusion maps

In the presented analysis, we used a diffusion map with sigma 8. Here, we calculated several diffusion maps on the same data with varying sigmas (5, 7, 8, 9, and 13) to demonstrate how we chose an appropriate sigma.

The transition probabilities between cells is their Euclidean distance in gene expression space (calculated on the highly variable genes), then transformed by a Gaussian function to prioritize transitions between cells that are very close. The sigma parameters is the standard deviation of the Gaussian used to transform the distances. A smaller sigma requires cells to be closer to each other in transcriptional space in order to be connected in the tree. An overly small sigma, however, will create disconnections, where all connections to some cells will become 0.

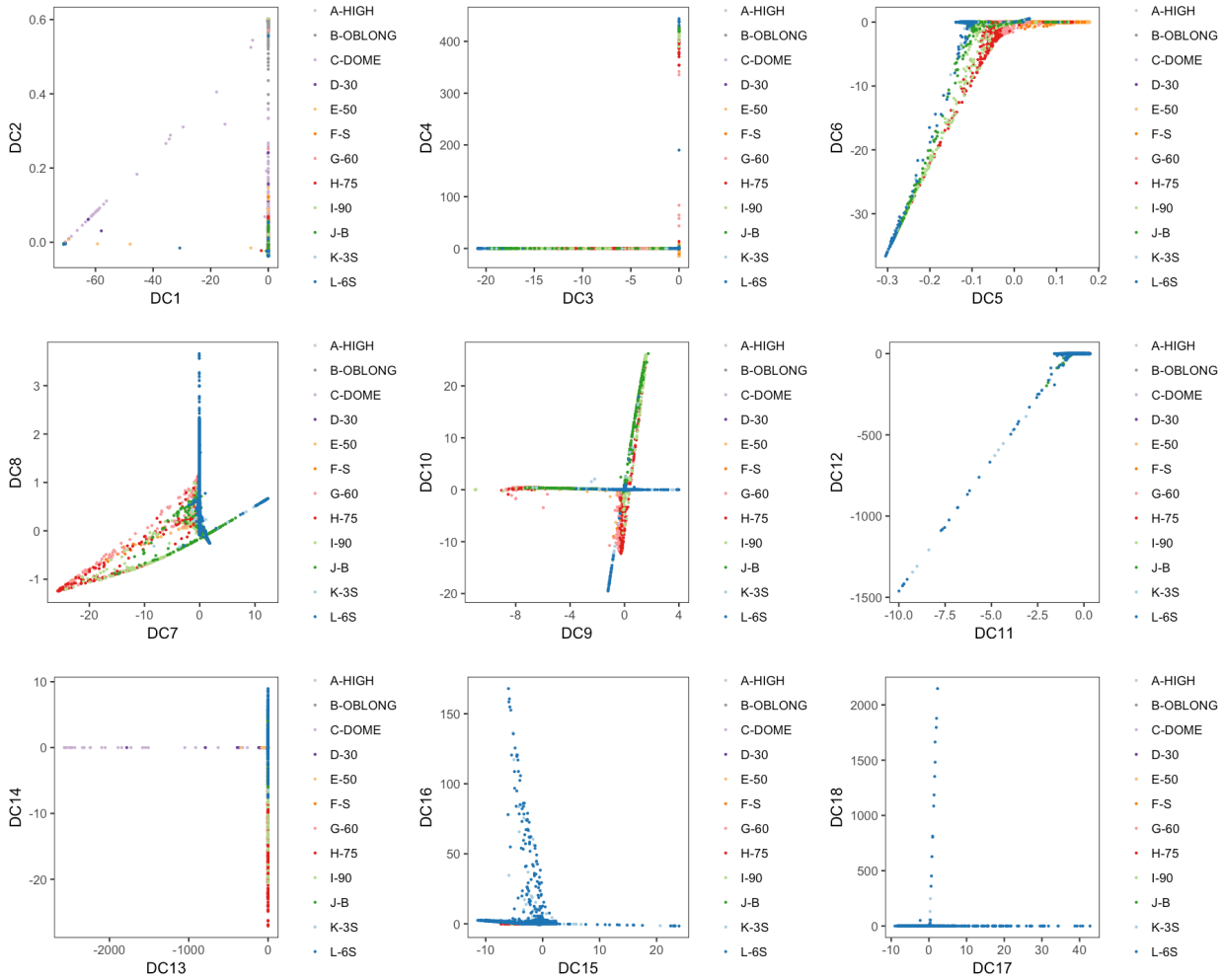
```
# Load calculated diffusion maps
dm.5 <- readRDS("dm/dm-5-2.0.6ep.rds")
dm.7 <- readRDS("dm/dm-7-2.0.6ep.rds")
dm.8 <- readRDS("dm/dm-8-2.0.6ep.rds")
dm.9 <- readRDS("dm/dm-9-2.0.6ep.rds")
dm.13 <- readRDS("dm/dm-13-2.0.6ep.rds")

# Add them to URD objects
object.dm5 <- importDM(object, dm.5)
object.dm7 <- importDM(object, dm.7)
object.dm8 <- importDM(object, dm.8)
object.dm9 <- importDM(object, dm.9)
object.dm13 <- importDM(object, dm.13)

# Clean up RAM.
rm(list = c("dm.5", "dm.7", "dm.8", "dm.9", "dm.13", "object"))
shhh <- gc()
```

Inspect diffusion maps

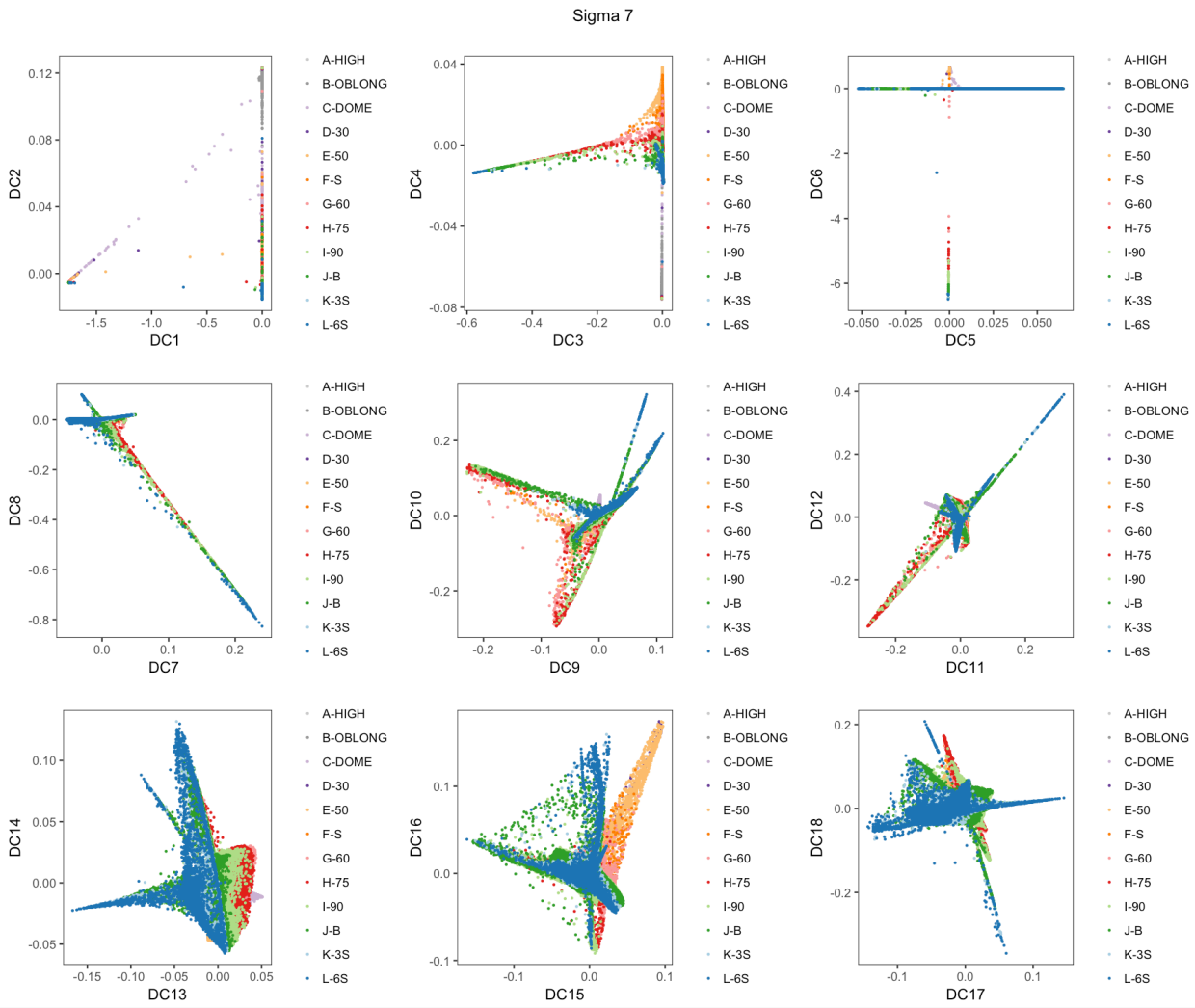
Choosing the correct sigma for the diffusion map and transition probabilities is critical. In general, we find it best to choose the smallest sigma possible that doesn't cause many disconnections in the data.

```
# Stage color palette
stage.colors <- c("#CCCCCC", RColorBrewer::brewer.pal(9, "Set1")[9], RColorBrewer::brewer.pal(12,
    "Paired")[c(9, 10, 7, 8, 5, 6, 3, 4, 1, 2)])

plotDimArray(object = object.dm5, reduction.use = "dm", dims.to.plot = 1:18, label = "stage.nice",
    plot.title = "", outer.title = "Sigma 5", discrete.colors = stage.colors)
```

Sigma 5

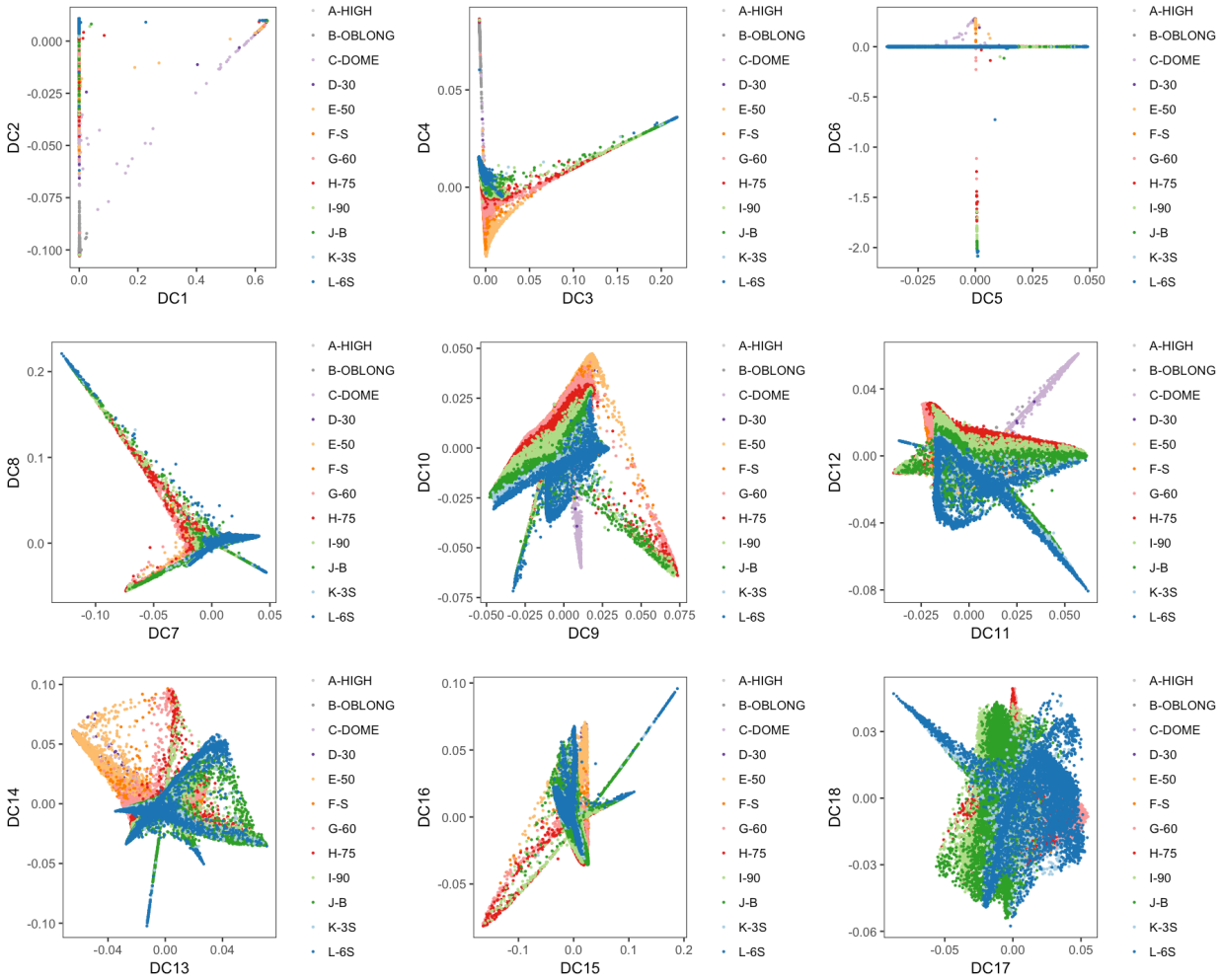



```
plotDimArray(object = object.dm7, reduction.use = "dm", dims.to.plot = 1:18, label = "stage.nice",
    plot.title = "", outer.title = "Sigma 7", discrete.colors = stage.colors)
```



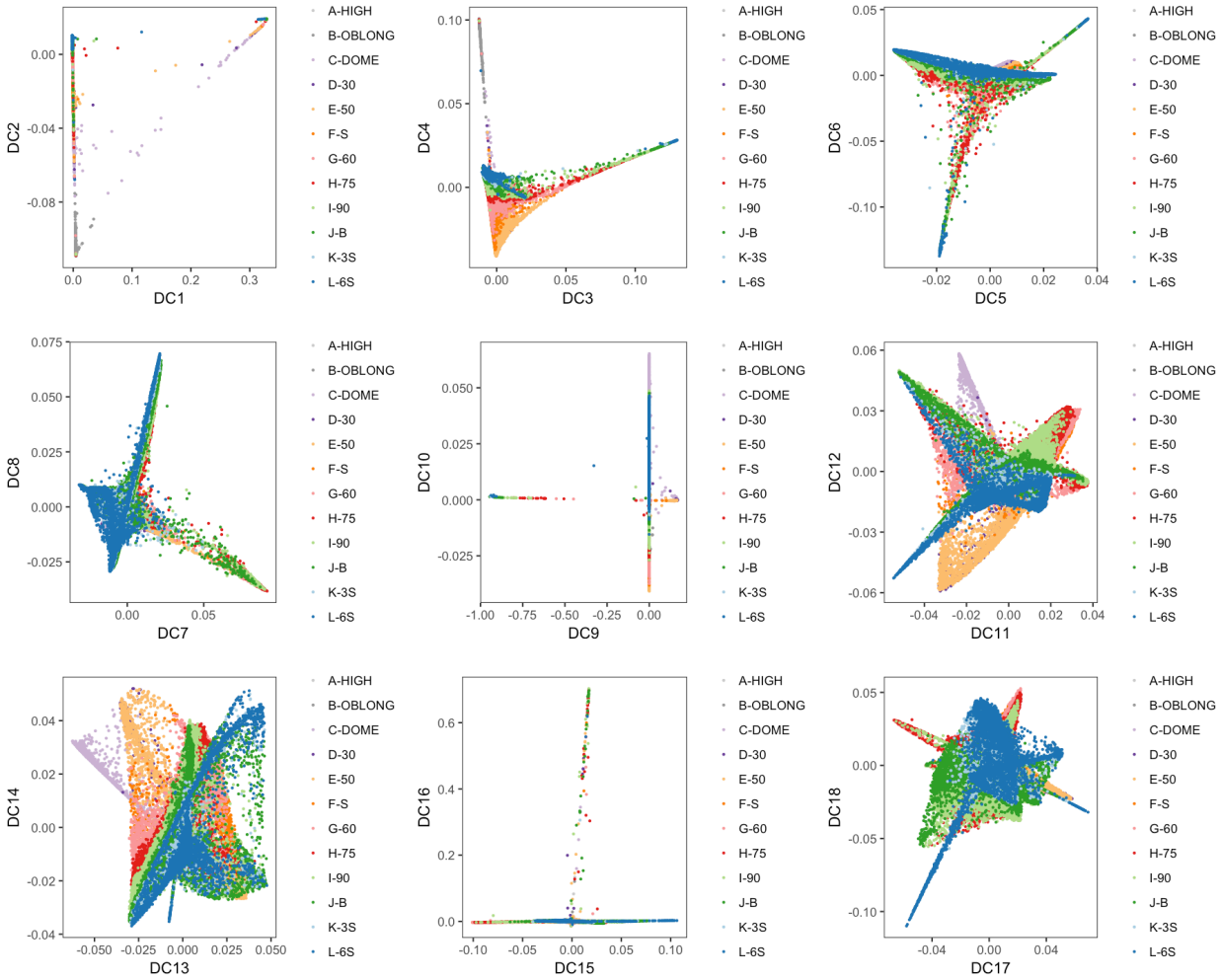

```
plotDimArray(object = object.dm8, reduction.use = "dm", dims.to.plot = 1:18, label = "stage.nice",
    plot.title = "", outer.title = "Sigma 8", discrete.colors = stage.colors)
```

Sigma 8

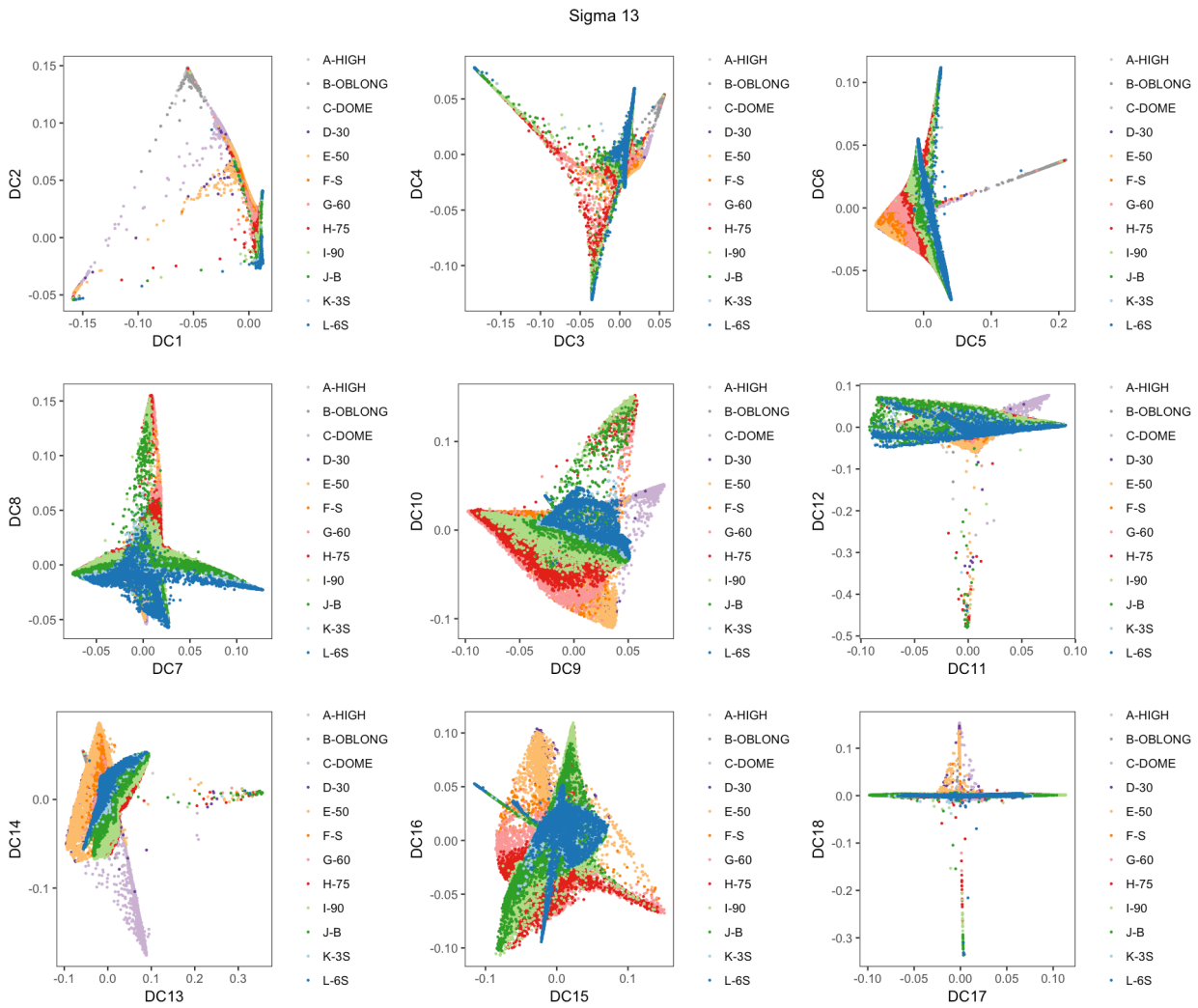



```
plotDimArray(object = object.dm9, reduction.use = "dm", dims.to.plot = 1:18, label = "stage.nice",
    plot.title = "", outer.title = "Sigma 9", discrete.colors = stage.colors)
```

Sigma 9




```
plotDimArray(object = object.dm13, reduction.use = "dm", dims.to.plot = 1:18, label = "stage.nice",
    plot.title = "", outer.title = "Sigma 13", discrete.colors = stage.colors)
```



In this case, we would choose sigma 8 as our preferred resolution, although sigmas 7 or 9 would also potentially work. Sigma 5 is too small and has many components that are essentially linear, while sigma 13 is too broad.

If you have DCs that define singleton cells, it means that outliers remain, and you should adjust the parameters of the outlier removal step prior to calculating the diffusion map (see Part 2).

Pseudotime

We used each of the above diffusion maps to determine pseudotime in our data, and compared them to the pseudotime defined by the diffusion map with sigma 8 that we used in our analysis.

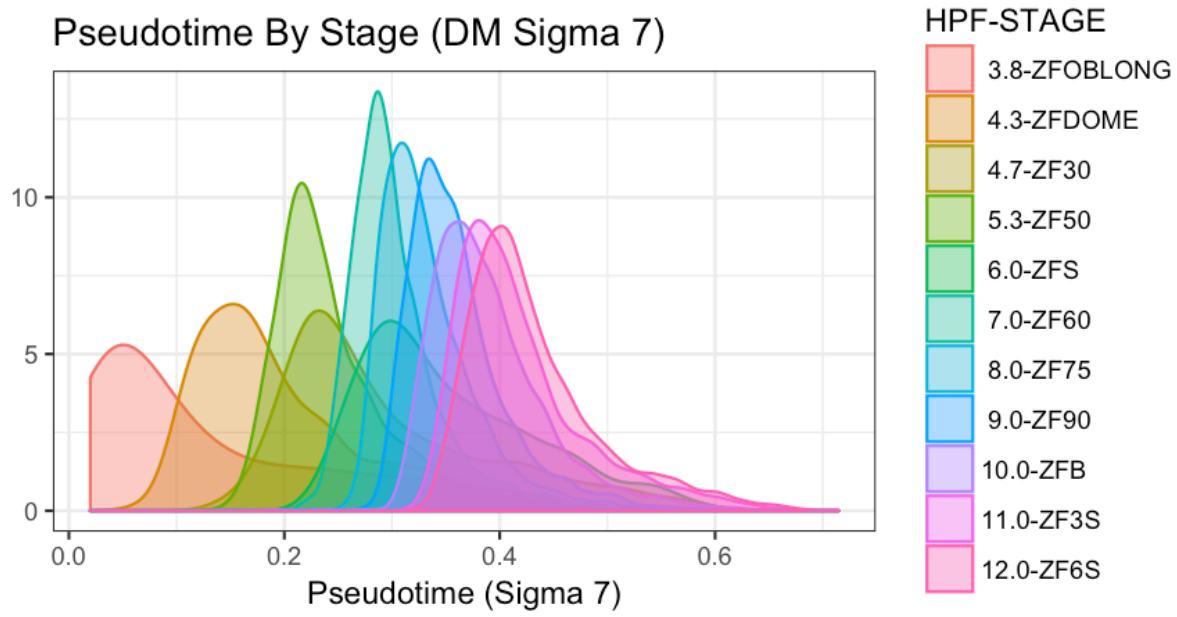
```
# Load floods
floods.dm5 <- lapply(list.files(path = "floods/", pattern = "flood-dm-5", full.names = T),
    readRDS)
floods.dm7 <- lapply(list.files(path = "floods/", pattern = "flood-dm-7", full.names = T),
    readRDS)
floods.dm8 <- lapply(list.files(path = "floods/", pattern = "flood-dm-8", full.names = T),
    readRDS)
floods.dm9 <- lapply(list.files(path = "floods/", pattern = "flood-dm-9", full.names = T),
    readRDS)
floods.dm13 <- lapply(list.files(path = "floods/", pattern = "flood-dm-13", full.names = T),
    readRDS)

# Process the floods
object.dm5 <- floodPseudotimeProcess(object.dm5, floods.dm5, floods.name = "pseudotime",
    max.frac.NA = 0.4, pseudotime.fun = mean, stability.div = 10)
object.dm7 <- floodPseudotimeProcess(object.dm7, floods.dm7, floods.name = "pseudotime",
```

```
    max.frac.NA = 0.4, pseudotime.fun = mean, stability.div = 10)
object.dm8 <- floodPseudotimeProcess(object.dm8, floods.dm8, floods.name = "pseudotime",
    max.frac.NA = 0.4, pseudotime.fun = mean, stability.div = 10)
object.dm9 <- floodPseudotimeProcess(object.dm9, floods.dm9, floods.name = "pseudotime",
    max.frac.NA = 0.4, pseudotime.fun = mean, stability.div = 10)
object.dm13 <- floodPseudotimeProcess(object.dm13, floods.dm13, floods.name = "pseudotime",
    max.frac.NA = 0.4, pseudotime.fun = mean, stability.div = 10)

pseudotime.compare <- data.frame(pseudotime.5 = object.dm5@pseudotime$pseudotime,
    pseudotime.7 = object.dm7@pseudotime$pseudotime, pseudotime.8 = object.dm8@pseudotime$pseudotime,
    pseudotime.9 = object.dm9@pseudotime$pseudotime, pseudotime.13 = object.dm13@pseudotime$pseudotime,
    row.names = rownames(object.dm8@pseudotime))
pseudotime.compare$STAGE <- apply(object.dm8@meta[rownames(pseudotime.compare), c("HPF",
    "STAGE")], 1, paste0, collapse = "-")
```

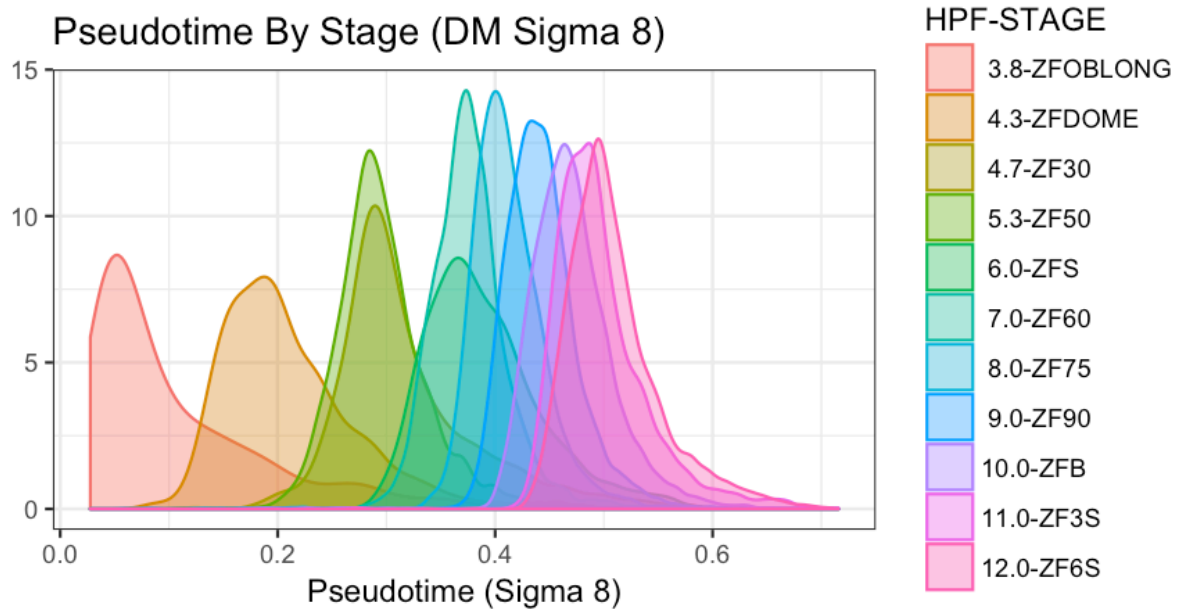
Pseudotime by stage

We first looked at the pseudotime distribution of each developmental stage. For sigma 5, the pseudotime calculation failed – only 5 cells in the data are assigned pseudotimes, because the graph is too poorly connected for the pseudotime simulations to visit most cells reliably. For sigmas 7-9, different developmental stages have distinct, but overlapping distributions of pseudotime. Sigma 7 is potentially slightly too small, as 30% and 50% epiboly stages occur in the wrong order. For an overly large sigmas (e.g. 13), most of the later stages collapse atop each other. We preferred sigma 8 to sigma 9 because 9 begins to show the collapse of the later stage distributions (at least compared to 8).

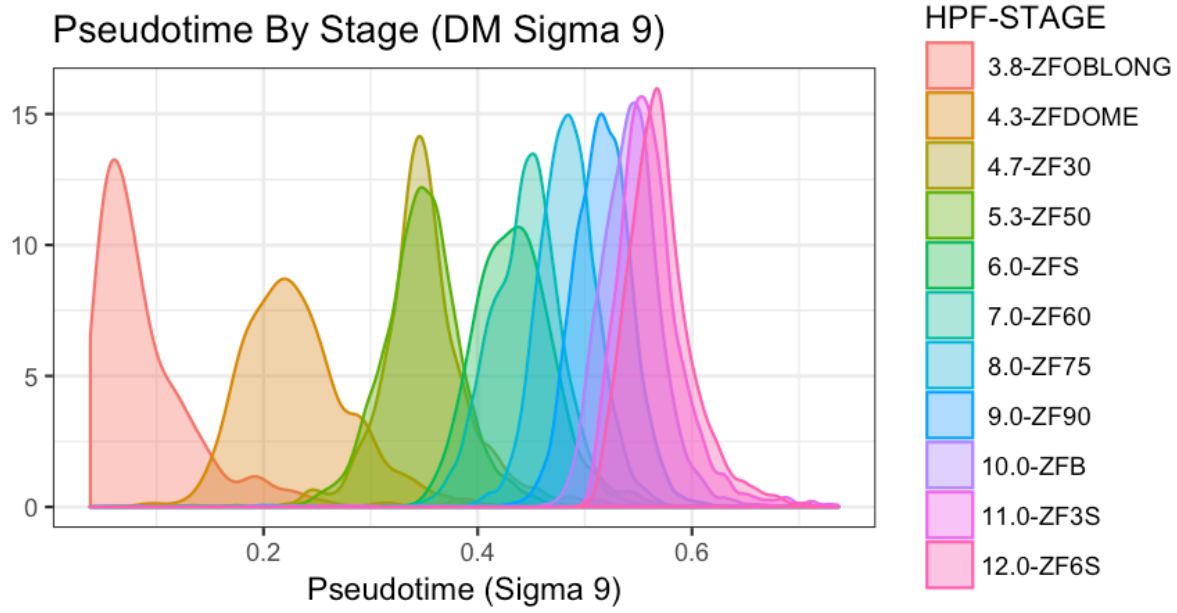
```
# Omit high stage -- they are all exactly 0 (since they were defined as the
# root), and it messes with the density distribution algorithm!
pseudotime.compare.nohigh <- pseudotime.compare[grep("ZFHIGH", rownames(pseudotime.compare),
    value = T, invert = T), ]

ggplot(pseudotime.compare.nohigh, aes(x = pseudotime.7, color = STAGE, fill = STAGE)) +
    geom_density(alpha = 0.4) + theme_bw() + labs(x = "Pseudotime (Sigma 7)", y = "",
    fill = "HPF-STAGE", color = "HPF-STAGE", title = "Pseudotime By Stage (DM Sigma 7)")
```



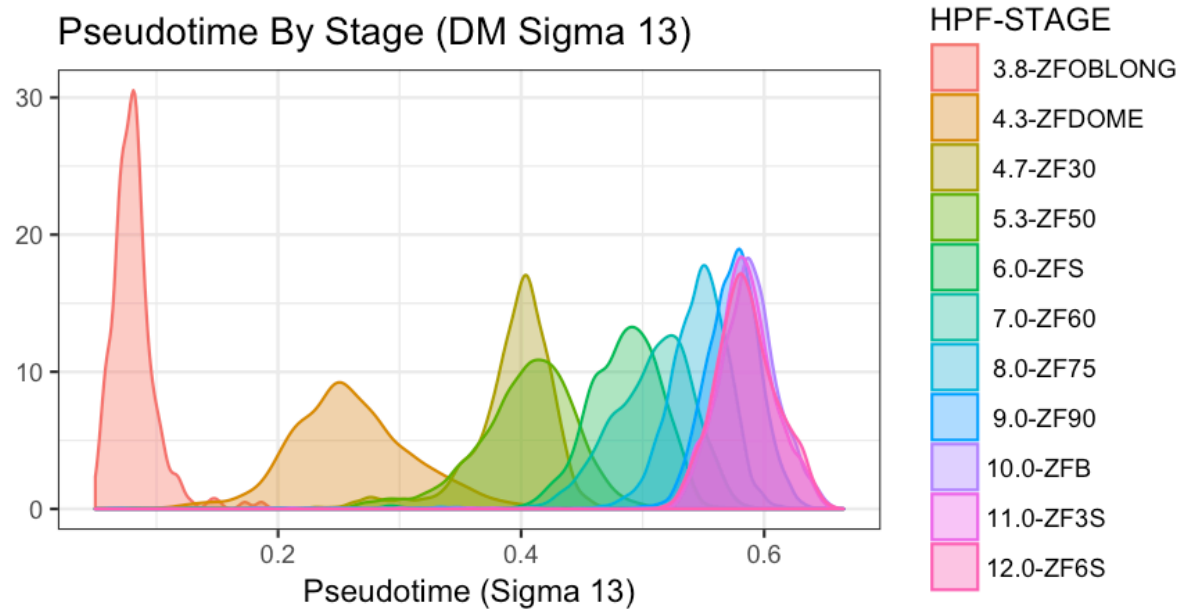

```
ggplot(pseudotime.compare.nohigh, aes(x = pseudotime.8, color = STAGE, fill = STAGE)) +
    geom_density(alpha = 0.4) + theme_bw() + labs(x = "Pseudotime (Sigma 8)", y = "",
    fill = "HPF-STAGE", color = "HPF-STAGE", title = "Pseudotime By Stage (DM Sigma 8)")
```




```
ggplot(pseudotime.compare.nohigh, aes(x = pseudotime.9, color = STAGE, fill = STAGE)) +
    geom_density(alpha = 0.4) + theme_bw() + labs(x = "Pseudotime (Sigma 9)", y = "",
    fill = "HPF-STAGE", color = "HPF-STAGE", title = "Pseudotime By Stage (DM Sigma 9)")
```



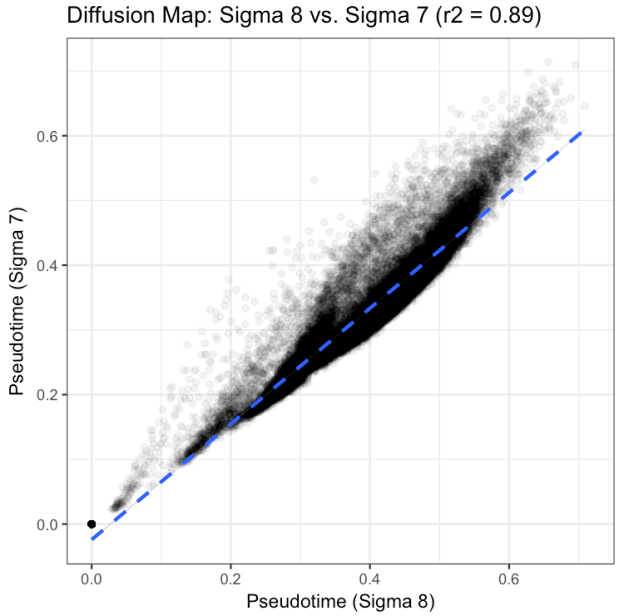

```
ggplot(pseudotime.compare.nohigh, aes(x = pseudotime.13, color = STAGE, fill = STAGE)) +
    geom_density(alpha = 0.4) + theme_bw() + labs(x = "Pseudotime (Sigma 13)", y = "",
    fill = "HPF-STAGE", color = "HPF-STAGE", title = "Pseudotime By Stage (DM Sigma 13)")
```



Compare determined pseudotimes

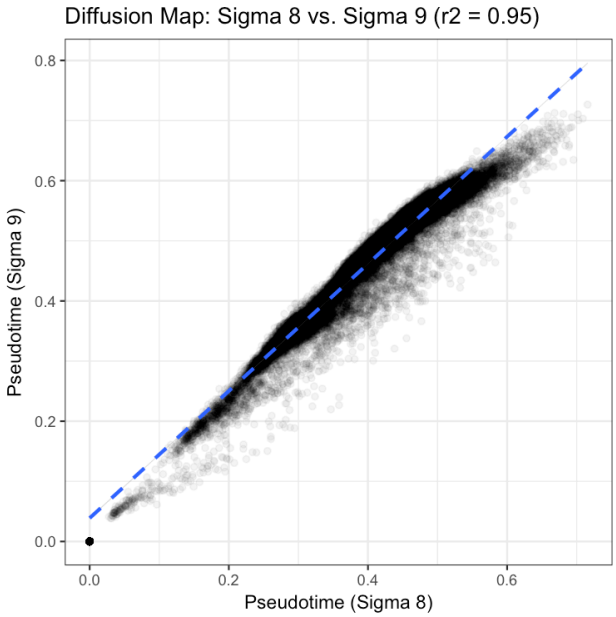
We next compared the pseudotime assignment of each cell between diffusion maps with different sigma. For sigmas that are similar to each other and to our chosen diffusion map (*i.e.* 7 and 9), the calculated pseudotimes are reasonably similar: nearly a linear transformation. For sigmas more distant from the optimal one (*e.g.* 13), can see that the overly connected diffusion map results in a pseudotime plateau. This mirrors the results we saw in the above plots of pseudotime by stage.

```
lm.8.7 <- lm(pseudotime.7 ~ pseudotime.8, data = pseudotime.compare)
r2.8.7 <- round(summary(lm.8.7)$r.squared, 2)
ggplot(pseudotime.compare, aes(x = pseudotime.8, y = pseudotime.7)) + geom_point(alpha = 0.05) +
    geom_smooth(method = "lm", lty = 2) + theme_bw() + labs(x = "Pseudotime (Sigma 8)",
    y = "Pseudotime (Sigma 7)", title = paste0("Diffusion Map: Sigma 8 vs. Sigma 7 (r2 = ",
        r2.8.7, ")"))
```

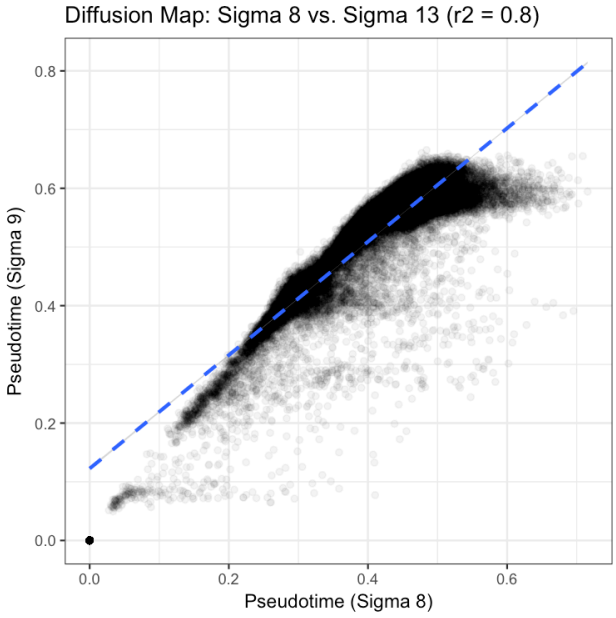



```
lm.8.9 <- lm(pseudotime.9 ~ pseudotime.8, data = pseudotime.compare)
r2.8.9 <- round(summary(lm.8.9)$r.squared, 2)
```

```
ggplot(pseudotime.compare, aes(x = pseudotime.8, y = pseudotime.9)) + geom_point(alpha = 0.05) +
    geom_smooth(method = "lm", lty = 2) + theme_bw() + labs(x = "Pseudotime (Sigma 8)",
    y = "Pseudotime (Sigma 9)", title = paste0("Diffusion Map: Sigma 8 vs. Sigma 9 (r2 = ",
        r2.8.9, ")"))
```




```
lm.8.13 <- lm(pseudotime.13 ~ pseudotime.8, data = pseudotime.compare)
r2.8.13 <- round(summary(lm.8.13)$r.squared, 2)
ggplot(pseudotime.compare, aes(x = pseudotime.8, y = pseudotime.13)) + geom_point(alpha = 0.05) +
    geom_smooth(method = "lm", lty = 2) + theme_bw() + labs(x = "Pseudotime (Sigma 8)",
    y = "Pseudotime (Sigma 9)", title = paste0("Diffusion Map: Sigma 8 vs. Sigma 13 (r2 = ",
        r2.8.13, ")"))
```



URD: Choosing Parameters - Biased Random Walk

```
library(URD)
library(gridExtra)  # grid.arrange
rgl::setupKnitr()

# Color schemes
fire.with.grey <- c("#CECECE", "#DDC998", RColorBrewer::brewer.pal(9, "YlOrRd")[3:9])
pond.with.grey <- c("#CECECE", "#CBDAC2", RColorBrewer::brewer.pal(9, "YlGnBu")[3:9])
```

Load previous saved object

```
object <- readRDS("obj/object_4_withTips.rds")
```

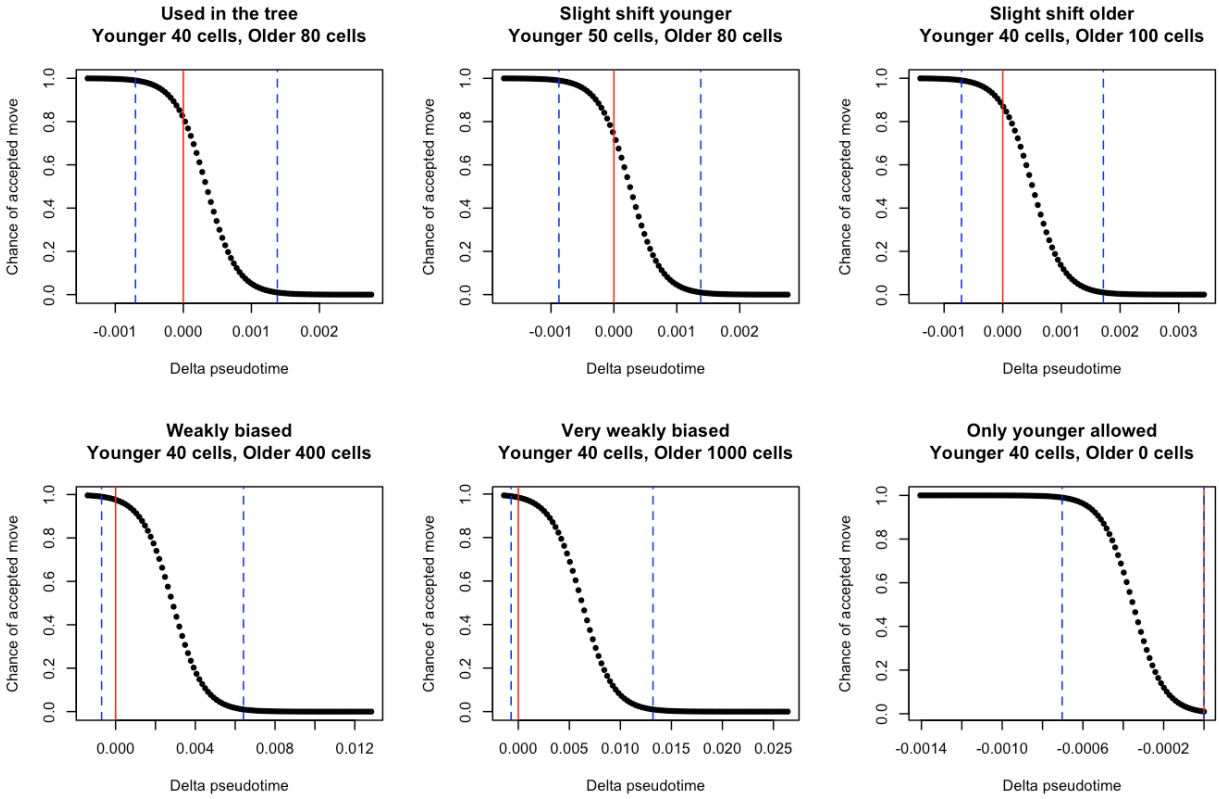
Modifying Random Walk Bias

Diffusion logistic settings

In order to perform the biased walks that determine the trajectories from each tip to the root, the transition probabilities must be biased, such that transitions to cells with younger pseudotime (i.e. closer to the root) are favored. We bias the random walks using a logistic function that provides a smooth curve that approaches 0 (totally prohibited) and 1 (probability unaltered). The parameters of the logistic are set in terms of a number of cells forward in pseudotime (across the entire dataset) and a number of cells backward in pseudotime (across the entire dataset) where the logistic approaches 1 and 0 respectively.

This parameter must be determined for each data set, as it is dependent on the number of cells present in the data set and likely the number of branches present. However, as we show below, the parameter is quite robust, and even extreme variations result in relatively small changes in the structure of the zebrafish tree.

```
par(mfrow = c(2, 3))
par(mar = c(5, 4, 5, 2))

diffusion.logistic <- pseudotimeDetermineLogistic(object, "pseudotime", optimal.cells.forward = 40,
    max.cells.back = 80, pseudotime.direction = "<", do.plot = T, print.values = F)
title("Used in the tree\nYounger 40 cells, Older 80 cells")

diffusion.logistic <- pseudotimeDetermineLogistic(object, "pseudotime", optimal.cells.forward = 50,
    max.cells.back = 80, pseudotime.direction = "<", do.plot = T, print.values = F)
title("Slight shift younger\nYounger 50 cells, Older 80 cells")

diffusion.logistic <- pseudotimeDetermineLogistic(object, "pseudotime", optimal.cells.forward = 40,
    max.cells.back = 100, pseudotime.direction = "<", do.plot = T, print.values = F)
title("Slight shift older\nYounger 40 cells, Older 100 cells")

diffusion.logistic <- pseudotimeDetermineLogistic(object, "pseudotime", optimal.cells.forward = 40,
    max.cells.back = 400, pseudotime.direction = "<", do.plot = T, print.values = F)
title("Weakly biased\nYounger 40 cells, Older 400 cells")

diffusion.logistic <- pseudotimeDetermineLogistic(object, "pseudotime", optimal.cells.forward = 40,
    max.cells.back = 1000, pseudotime.direction = "<", do.plot = T, print.values = F)
title("Very weakly biased\nYounger 40 cells, Older 1000 cells")

diffusion.logistic <- pseudotimeDetermineLogistic(object, "pseudotime", optimal.cells.forward = 40,
    max.cells.back = 0, pseudotime.direction = "<", do.plot = T, print.values = F)
title("Only younger allowed\nYounger 40 cells, Older 0 cells")
```




```
object.40.0 <- object
object.40.0@diff.data <- readRDS("alt/diff.data/diffdata-dm-8-tm-40-0.rds")
object.40.0@pseudotime <- readRDS("alt/diff.data/pseudotime-dm-8-tm-40-0.rds")
object.40.0@tree <- readRDS("alt/tree/tree-dm-8-tm-40-0.rds")

object.40.100 <- object
object.40.100@diff.data <- readRDS("alt/diff.data/diffdata-dm-8-tm-40-100.rds")
object.40.100@pseudotime <- readRDS("alt/diff.data/pseudotime-dm-8-tm-40-100.rds")
object.40.100@tree <- readRDS("alt/tree/tree-dm-8-tm-40-100.rds")

# Fixed a layout bug since running on the cluster so need to redo last steps
# of layout.
object.40.100 <- treeLayoutElaborate(object.40.100)
object.40.100 <- treeLayoutCells(object.40.100, pseudotime = "pseudotime")

object.40.400 <- object
object.40.400@diff.data <- readRDS("alt/diff.data/diffdata-dm-8-tm-40-400.rds")
object.40.400@pseudotime <- readRDS("alt/diff.data/pseudotime-dm-8-tm-40-400.rds")
object.40.400@tree <- readRDS("alt/tree/tree-dm-8-tm-40-400.rds")

object.40.1000 <- object
object.40.1000@diff.data <- readRDS("alt/diff.data/diffdata-dm-8-tm-40-1000.rds")
object.40.1000@pseudotime <- readRDS("alt/diff.data/pseudotime-dm-8-tm-40-1000.rds")
object.40.1000@tree <- readRDS("alt/tree/tree-dm-8-tm-40-1000.rds")

object.50.80 <- object
object.50.80@diff.data <- readRDS("alt/diff.data/diffdata-dm-8-tm-50-80.rds")
object.50.80@pseudotime <- readRDS("alt/diff.data/pseudotime-dm-8-tm-50-80.rds")
object.50.80@tree <- readRDS("alt/tree/tree-dm-8-tm-50-80.rds")

# Fixed a layout bug since running on the cluster so need to redo last steps
# of layout.
object.50.80 <- treeLayoutElaborate(object.50.80)
object.50.80 <- treeLayoutCells(object.50.80, pseudotime = "pseudotime")

object.dm7 <- object
object.dm7@diff.data <- readRDS("alt/diff.data/diffdata-dm-7-tm-40-80.rds")
```

```
object.dm7@pseudotime <- readRDS("alt/diff.data/pseudotime-dm-7-tm-40-80.rds")
object.dm7@tree <- readRDS("alt/tree/tree-dm-7-tm-40-80.rds")

object.dm9 <- object
object.dm9@diff.data <- readRDS("alt/diff.data/diffdata-dm-9-tm-40-80.rds")
object.dm9@pseudotime <- readRDS("alt/diff.data/pseudotime-dm-9-tm-40-80.rds")
object.dm9@tree <- readRDS("alt/tree/tree-dm-9-tm-40-80.rds")
```

Comparing visitation frequencies

We compared the visitation structure at the axial mesoderm branchpoint given different biased random walk settings (including no bias at all). We plotted both the segment assignment, as well as the visitation frequency of cells from the notochord and prechordal plate tips. Cell arrangement is based on the position of cells in 3 components of the diffusion map. These plots are cropped to cells that were part of the early blastoderm or the axial mesoderm lineage, and include cells that are general early blastoderm progenitors, the axial mesoderm progenitors, or the notochord or prechordal plate progenitors. While it is clear that biasing the random walks is important (as the unbiased random walks lead to visitation of both populations from either tip), the particular settings of the bias make only small changes to the visitation frequency near the branchpoint and the final placement of the branchpoint assignment.

```
## Incorporate unbiased walks from tip 29 = PCP Load unbiased walks
unbiased.walks.29 <- unlist(lapply(list.files(path = "walks/dm-8-unbiased/",
    pattern = "-29-", full.names = T), readRDS), recursive = F)
# How many actually completed?  75% completed, ~35k successful Load them
# into the object
object.unbiased <- processRandomWalks(object, walks = unbiased.walks.29, walks.name = "29.unbiased",
    n.subsample = 1, verbose = F)
# Clear up RAM because the unbiased walks lists are BIG.
rm(unbiased.walks.29)
shhh <- gc(verbose = F)

## Repeat for unbiased walks from tip 32 = Notochord
unbiased.walks.32 <- unlist(lapply(list.files(path = "walks/dm-8-unbiased/",
    pattern = "-32-", full.names = T), readRDS), recursive = F)
# 75% completed, ~35k successful
object.unbiased <- processRandomWalks(object.unbiased, walks = unbiased.walks.32,
    walks.name = "32.unbiased", n.subsample = 1, verbose = F)
rm(unbiased.walks.32)
shhh <- gc()

# Move diffusion map to unbiased walk object
object.unbiased@dm <- object@dm

object <- readRDS("obj/object_5_withWalks.rds")
object.tree <- readRDS("obj/object_6_tree.rds")

color.lim <- range(unlist(c(object@diff.data[, c("visitfreq.log.29", "visitfreq.log.32")],
    object.unbiased@diff.data[, c("visitfreq.log.29.unbiased", "visitfreq.log.32.unbiased")],
    object.40.0@diff.data[, c("visitfreq.log.29", "visitfreq.log.32")], object.40.1000@diff.data[,
        c("visitfreq.log.29", "visitfreq.log.32")])))

cells.to.3d.plot <- cellsInCluster(object.tree, "segment", c("81", "79", "29",
    "32"))

# Load plotDim3D orientation into your objects
object@plot.3d <- readRDS("obj/object_6_tree_axialmesoplot3d.rds")
object.tree@plot.3d <- readRDS("obj/object_6_tree_axialmesoplot3d.rds")
object.unbiased@plot.3d <- readRDS("obj/object_6_tree_axialmesoplot3d.rds")
object.40.0@plot.3d <- readRDS("obj/object_6_tree_axialmesoplot3d.rds")
object.40.1000@plot.3d <- readRDS("obj/object_6_tree_axialmesoplot3d.rds")

# Recreate 'segment' group identity for 3D plots later
object.40.0@group.ids$segment <- "99"
for (segment in c("29", "32", "79", "81")) {
    object.40.0@group.ids[object.40.0@tree$cells.in.segment[[segment]], "segment"] <- segment
}
object.40.1000@group.ids$segment <- "99"
for (segment in c("29", "32", "79", "81")) {
    object.40.1000@group.ids[object.40.1000@tree$cells.in.segment[[segment]],
```

```
        "segment"] <- segment
}
```

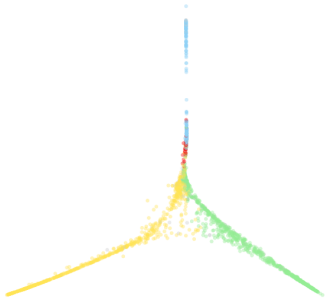
Segment assignment

Here, the segment assignment is plotted for three biased random walk conditions. These plots reveal that there are very slight changes in the placement of the notochord-prechordal plate branchpoint depending on the random walk parameters – when more lateral and backward movement is allowed (such as in 40F/1000B), the axial mesoderm common progenitor segment persists slightly later before splitting into the notochord and prechordal plate, and the converse occurs when less lateral or backward movement is allowed (such as in 40F/0B). Colors: blue (early blastoderm progenitors), red (axial mesoderm progenitors), green (prechordal plate progenitors), yellow (notochord progenitors).

```
plotDim3D(object.tree, view = "axialmeso", label = "segment", cells = cells.to.3d.plot,
    bounding.box = F, discrete.colors = c("#FFE354", "#93EC93", "#FF0000", "#8CD0F5"),
    size = 6, alpha = 0.4, title = "Segment identities: 40F/80B", title.line = -35)
rgl::points3d(x = 0, y = 0, z = 0, col = "white", alpha = 0.1, size = 1)
```

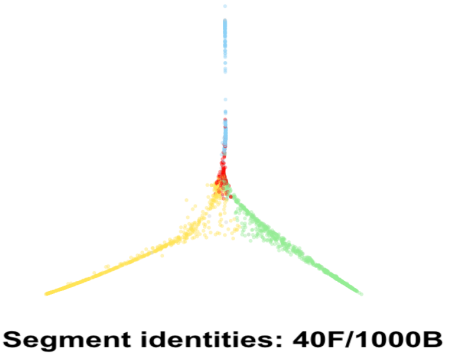

Segment identities: 40F/80B

```
plotDim3D(object.40.0, view = "axialmeso", label = "segment", cells = cells.to.3d.plot,
    bounding.box = F, discrete.colors = c("#FFE354", "#93EC93", "#FF0000", "#8CD0F5",
        "#CECECE"), size = 6, alpha = 0.4, title = "Segment identities: 40F/0B",
    title.line = -35)
rgl::points3d(x = 0, y = 0, z = 0, col = "white", alpha = 0.1, size = 1)
```



Segment identities: 40F/0B

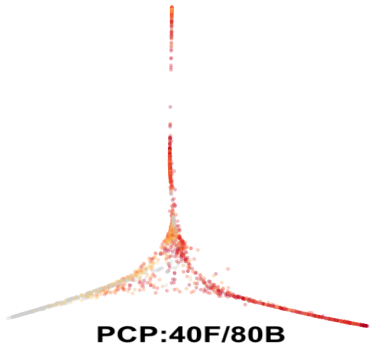
```
plotDim3D(object.40.1000, view = "axialmeso", label = "segment", cells = cells.to.3d.plot,
    bounding.box = F, discrete.colors = c("#FFE354", "#93EC93", "#FF0000", "#8CD0F5",
        "#CECECE"), size = 6, alpha = 0.4, title = "Segment identities: 40F/1000B",
    title.line = -35)
rgl::points3d(x = 0, y = 0, z = 0, col = "white", alpha = 0.1, size = 1)
```



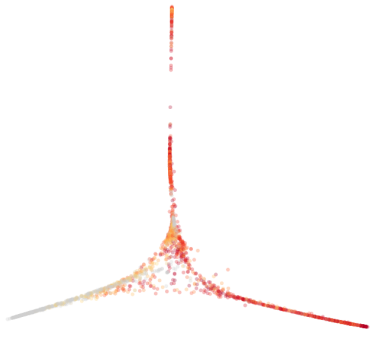
Prechordal Plate Walks

Here, visitation frequency from the prechordal plate tip is plotted. Visitation frequency is only slightly affected by the particular parameters used for biasing the transition matrix, however, the final "unbiased" condition shows that the biasing step is absolutely critical.

```
plotDim3D(object, label = "visitfreq.log.29", view = "axialmeso2", dim.1 = 3,
    dim.2 = 8, dim.3 = 1, cells = cells.to.3d.plot, bounding.box = F, continuous.colors = fire.with.grey,
    continuous.color.limits = color.lim, size = 6, alpha = 0.3, title = "PCP:40F/80B",
    title.line = -35)
rgl::points3d(x = 0, y = 0, z = 0, col = "white", alpha = 0.1, size = 1)
```

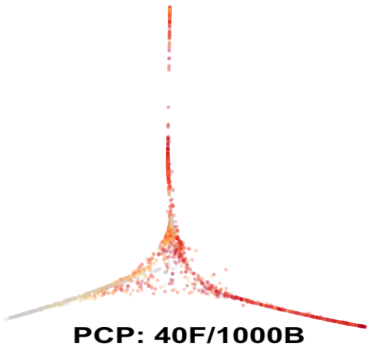



```
plotDim3D(object.40.0, label = "visitfreq.log.29", view = "axialmeso2", dim.1 = 3,
    dim.2 = 8, dim.3 = 1, cells = cells.to.3d.plot, bounding.box = F, continuous.colors = fire.with.grey,
    continuous.color.limits = color.lim, size = 6, alpha = 0.3, title = "PCP: 40F/0B",
    title.line = -35)
rgl::points3d(x = 0, y = 0, z = 0, col = "white", alpha = 0.1, size = 1)
```

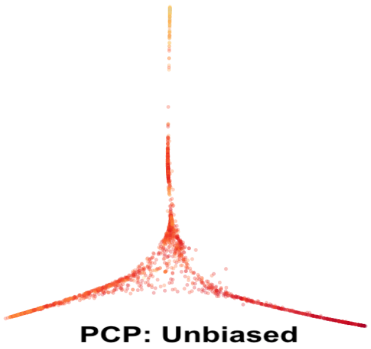


```
plotDim3D(object.40.1000, label = "visitfreq.log.29", view = "axialmeso2", dim.1 = 3,
    dim.2 = 8, dim.3 = 1, cells = cells.to.3d.plot, bounding.box = F, continuous.colors = fire.with.grey,
```

```
    continuous.color.limits = color.lim, size = 6, alpha = 0.3, title = "PCP: 40F/1000B",
    title.line = -35)
rgl::points3d(x = 0, y = 0, z = 0, col = "white", alpha = 0.1, size = 1)
```



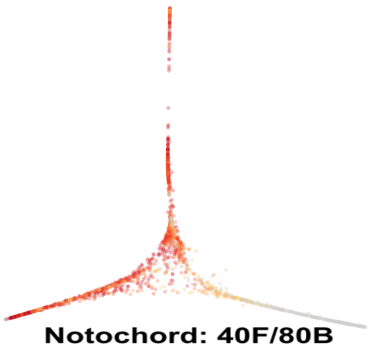

```
plotDim3D(object.unbiased, label = "visitfreq.log.29.unbiased", view = "axialmeso2",
    dim.1 = 3, dim.2 = 8, dim.3 = 1, cells = cells.to.3d.plot, bounding.box = F,
    continuous.colors = fire.with.grey, continuous.color.limits = color.lim,
    size = 6, alpha = 0.3, title = "PCP: Unbiased", title.line = -35)
rgl::points3d(x = 0, y = 0, z = 0, col = "white", alpha = 0.1, size = 1)
```



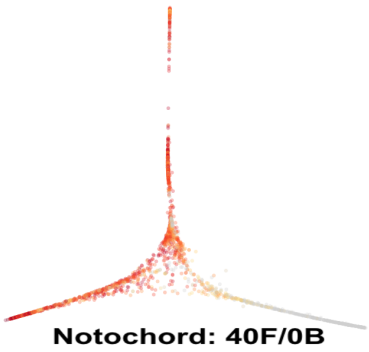
Notochord Walks

Here, visitation frequency from the notochord tip is plotted. Visitation frequency is only slightly affected by the particular parameters used for biasing the transition matrix, however, the final "unbiased" condition shows that the biasing step is absolutely critical.

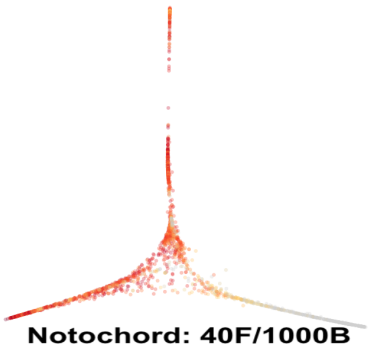
```
plotDim3D(object, label = "visitfreq.log.32", view = "axialmeso2", dim.1 = 3,
    dim.2 = 8, dim.3 = 1, cells = cells.to.3d.plot, bounding.box = F, continuous.colors = fire.with.grey,
    continuous.color.limits = color.lim, size = 6, alpha = 0.3, title = "Notochord: 40F/80B",
    title.line = -35)
rgl::points3d(x = 0, y = 0, z = 0, col = "white", alpha = 0.1, size = 1)
```



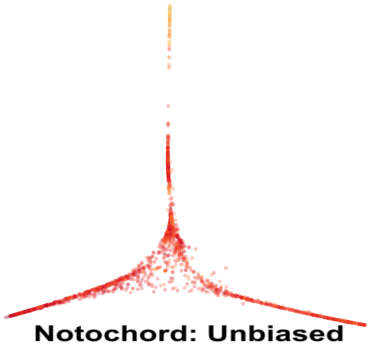

```
plotDim3D(object.40.0, label = "visitfreq.log.32", view = "axialmeso2", dim.1 = 3,
    dim.2 = 8, dim.3 = 1, cells = cells.to.3d.plot, bounding.box = F, continuous.colors = fire.with.grey,
    continuous.color.limits = color.lim, size = 6, alpha = 0.3, title = "Notochord: 40F/0B",
    title.line = -35)
rgl::points3d(x = 0, y = 0, z = 0, col = "white", alpha = 0.1, size = 1)
```




```
plotDim3D(object.40.1000, label = "visitfreq.log.32", view = "axialmeso2", dim.1 = 3,
    dim.2 = 8, dim.3 = 1, cells = cells.to.3d.plot, bounding.box = F, continuous.colors = fire.with.grey,
    continuous.color.limits = color.lim, size = 6, alpha = 0.3, title = "Notochord: 40F/1000B",
    title.line = -35)
rgl::points3d(x = 0, y = 0, z = 0, col = "white", alpha = 0.1, size = 1)
```




```
plotDim3D(object.unbiased, label = "visitfreq.log.32.unbiased", view = "axialmeso2",
    dim.1 = 3, dim.2 = 8, dim.3 = 1, cells = cells.to.3d.plot, bounding.box = F,
    continuous.colors = fire.with.grey, continuous.color.limits = color.lim,
    title = "Notochord: Unbiased", title.line = -35, size = 6, alpha = 0.3)
rgl::points3d(x = 0, y = 0, z = 0, col = "white", alpha = 0.1, size = 1)
```

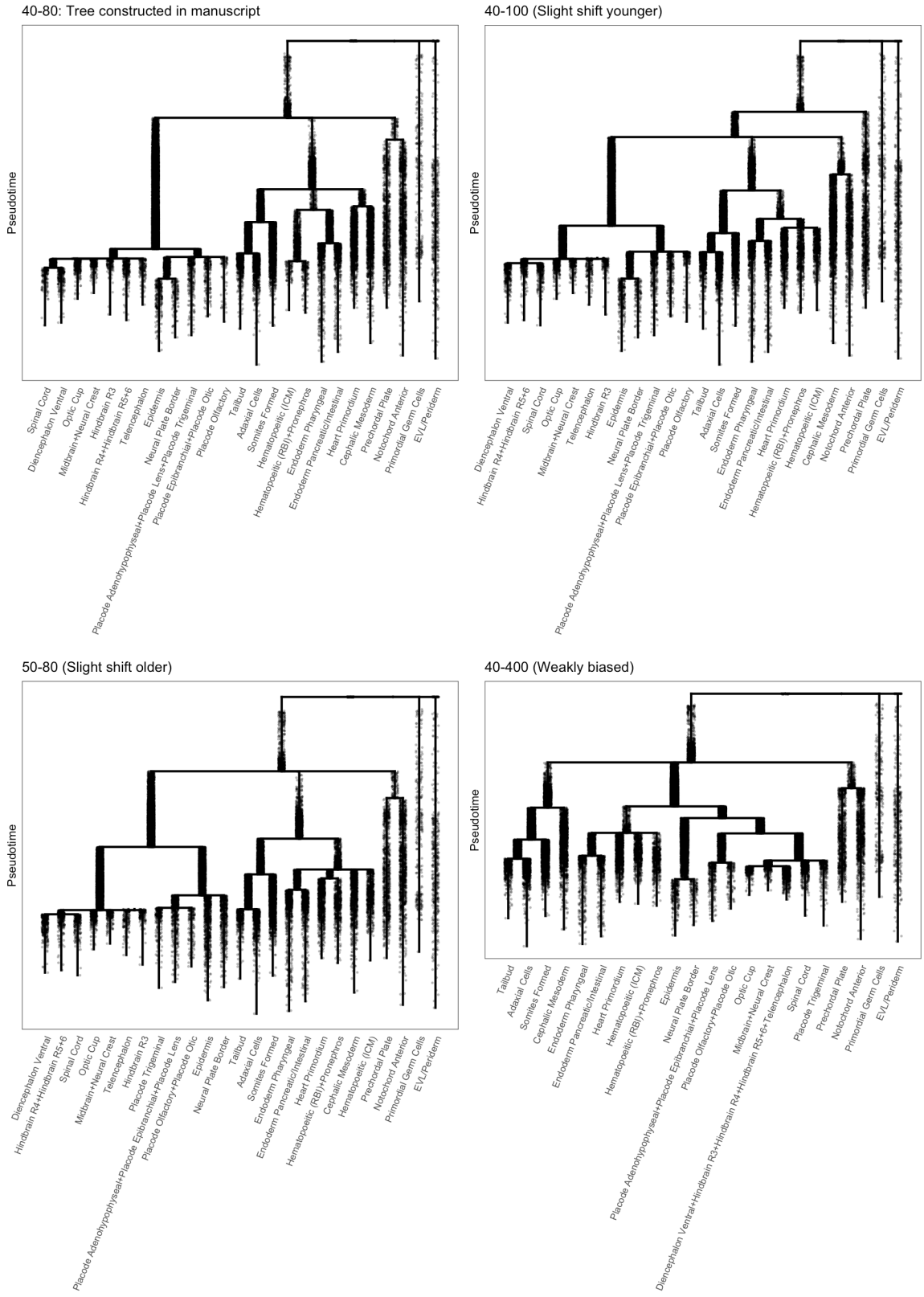


Compare Resultant Tree Structures

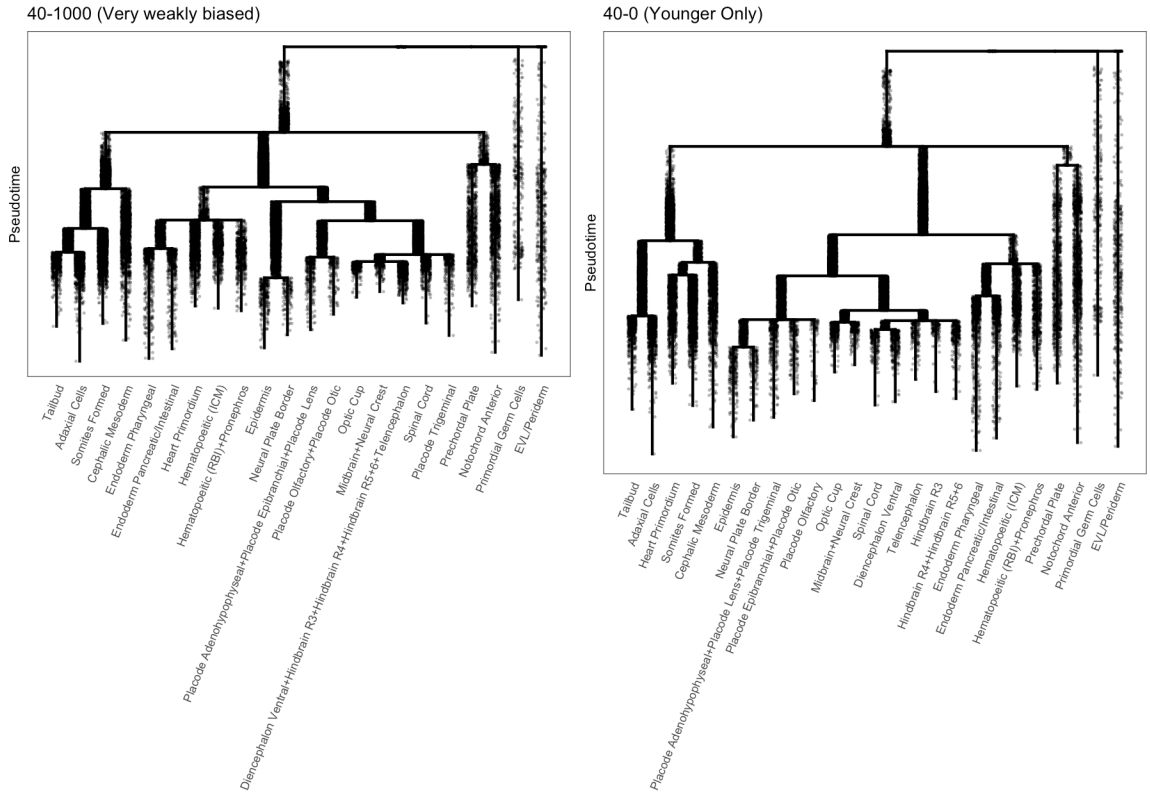
Much of the tree structure is preserved across all of these trees, though there is some variability.

The "slight shift younger" results in no change in the overall structure of the tree, and the "slight shift older" results only in a slight change of the structure of the axial mesoderm (the cephalic mesoderm joins the axial mesoderm, and the prechordal plate begins to separate earlier than the notochord). The "weakly biased" parameters also result in only a small change in the structure of the tree (the same rearrangement of the axial mesoderm, and a slightly earlier branchpoint of the preplacodal ectoderm from the remainder of the non-neural ectoderm). The "very weakly biased" parameters, introduce more changes (now the paraxial mesoderm splits earlier, along with the axial mesoderm, and the remainder of the mesendoderm branches from the ectoderm later). However, even under these extreme parameters, most aspects of the tree are reproduced and recapitulate known embryology. Finally, the "younger only" parameters result in a similar tree structure as the "very weakly biased" parameters.

```
# Move the object with built tree over to the main object slot.
object <- object.tree
rm(object.tree)
# Automatically name the tips, so they will match the other trees
tip.names <- unique(object@group.ids[, c("ZF6S-Cluster", "ZF6S-Cluster-Num")])
tip.names <- tip.names[complete.cases(tip.names), ]
object <- nameSegments(object, segments = tip.names$`ZF6S-Cluster-Num`, segment.names = tip.names$`ZF6S-Cluster`)

tree.plots.1 <- list(plotTree(object, title = "40-80: Tree constructed in manuscript"),
    plotTree(object.40.100, title = "40-100 (Slight shift younger)"), plotTree(object.50.80,
        title = "50-80 (Slight shift older)"), plotTree(object.40.1000, title = "40-400 (Weakly biased)"))
tree.plots.2 <- list(plotTree(object.40.1000, title = "40-1000 (Very weakly biased)"),
    plotTree(object.40.0, title = "40-0 (Younger Only)"))

grid.arrange(grobs = tree.plots.1, ncol = 2)
```




```
grid.arrange(grobs = tree.plots.2, ncol = 2)
```



Compare changes in trajectory membership across entire tree

We quantified the percentage of the assignments (of cell to trajectory) that changed in each tree. Each cell was determined as "in" or "not-in" each trajectory, and the percentage of those assignments that changed for each alternative tree relative to the tree presented in the paper was determined.

```
alts <- c("object.40.100", "object.50.80", "object.40.400", "object.40.1000",
    "object.40.0")

tree.change <- lapply(alts, function(alt) {

    object.alt <- get(alt)

    # Some tips combined immediately, but this could be different between trees.
    # Create a data frame for each original tip to the terminal segment that it
    # belongs to in each resultant tree.

    tree.terminal.segs <- data.frame(original.tip = object@tree$tips, row.names = object@tree$tips,
        stringsAsFactors = F)

    tree.terminal.segs[, 2:3] <- t(apply(tree.terminal.segs, 1, function(ot) {
        ot <- ot[1]
        if (ot %in% segTerminal(object)) {
            seg.1 <- ot
        } else {
            t <- ot
            while (!(t %in% segTerminal(object))) {
                t <- object@tree$segment.joins.initial[c(which(object@tree$segment.joins.initial$child.1 ==
                  t), which(object@tree$segment.joins.initial$child.2 == t)),
                  "parent"]
            }
            seg.1 <- t
        }
        if (ot %in% segTerminal(object.alt)) {
            seg.2 <- ot
        } else {
```

```
            t <- ot
            while (!(t %in% segTerminal(object.alt))) {
                t <- object.alt@tree$segment.joins.initial[c(which(object.alt@tree$segment.joins.initial$child.1 ==
                  t), which(object.alt@tree$segment.joins.initial$child.2 ==
                  t)), "parent"]
            }
            seg.2 <- t
        }
        return(c(seg.1, seg.2))
    }))

    # Now, get the cells in the tree overall for both trees so you only consider
    # those.
    cells.in.tree.original <- unique(unlist(object@tree$cells.in.segment))
    cells.in.tree.alt <- unique(unlist(object.alt@tree$cells.in.segment))
    cells.in.trees <- intersect(cells.in.tree.original, cells.in.tree.alt)

    # Now, for each lineage, figure out the number of cells in that lineage for
    # each tree
    tree.terminal.segs[, c(4:7)] <- t(apply(tree.terminal.segs, 1, function(ts) {
        cells.orig <- intersect(cellsAlongLineage(object, segments = ts[2],
            remove.root = F), cells.in.trees)
        cells.alt <- intersect(cellsAlongLineage(object.alt, segments = ts[3],
            remove.root = F), cells.in.trees)
        cells.both <- length(intersect(cells.orig, cells.alt))
        cells.orig.only <- length(setdiff(cells.orig, cells.alt))
        cells.alt.only <- length(setdiff(cells.alt, cells.orig))
        cells.neither <- length(cells.in.trees) - (cells.both + cells.orig.only +
            cells.alt.only)
        return(c(cells.both, cells.orig.only, cells.alt.only, cells.neither))
    }))

    names(tree.terminal.segs) <- c("tip.run", "tip.orig", "tip.alt", "cells.both",
        "cells.orig", "cells.alt", "cells.neither")

    return(tree.terminal.segs)
})
names(tree.change) <- alts

# Figure out how much each changed.

tree.changed.overall <- unlist(lapply(tree.change, function(tc) {
    tcsum <- apply(tc[, 4:7], 2, sum)
    changed <- round(((tcsum[2] + tcsum[3])/sum(tcsum)) * 100, digits = 2)
    return(changed)
}))

tree.changed.overall.present <- paste0(tree.changed.overall, "%")
names(tree.changed.overall.present) <- c("40-100", "50-80", "40-400", "40-1000",
    "40-0")

tree.changed.overall.present
```

```
##   40-100    50-80   40-400  40-1000     40-0
##  "9.05%" "10.91%"  "9.43%" "14.02%" "11.07%"
```

Overall, the tree assignments are fairly robust to the parameter used for biasing the random walks, as only about 10% of cells' assignments change. The most extreme change is in the "very weakly biased" parameters (40-1000: 14.02%), as expected from the more dramatic changes in the structure of the tree.

Compare changes in trajectory membership for each trajectory

We also wondered if some parts of the tree were more sensitive to others than the parameters of the reconstruction. So, we quantified the percentage change in each lineage, given the different parameters.

```
tree.changed.by.lineage <- as.data.frame(lapply(tree.change, function(tc) {
    apply(tc[, 4:7], 1, function(tcr) {
        changed <- round((tcr[2] + tcr[3])/sum(tcr[1:4]) * 100, digits = 2)
    })
}))
```

```
tip.names <- unique(object@group.ids[, c("ZF6S-Cluster-Num", "ZF6S-Cluster")])
tip.names <- tip.names[complete.cases(tip.names), ]
rownames(tip.names) <- tip.names[, "ZF6S-Cluster-Num"]
tree.changed.by.lineage$population <- tip.names[rownames(tree.changed.by.lineage),
    "ZF6S-Cluster"]

names(tree.changed.by.lineage) <- c("40-100", "50-80", "40-400", "40-1000",
    "40-0", "Population")

tree.changed.by.lineage
```

```
##    40-100 50-80 40-400 40-1000  40-0                    Population
## 1   14.16 20.35  13.89   19.74 14.84                     Epidermis
## 2   10.03 13.23  10.31   14.56 11.86                     Optic Cup
## 3    6.27  4.02   5.51    5.30  4.26                       Tailbud
## 5    6.98  4.69   7.03   18.14  4.59               Heart Primordium
## 8   10.46 13.65  11.23   15.24 11.72                    Spinal Cord
## 10  14.16 20.77  13.90   19.73 14.84           Neural Plate Border
## 12   6.76  3.92   5.76    5.81  8.17                Somites Formed
## 16  10.10 13.28  10.24   16.14 11.53                Telencephalon
## 17   7.33  6.03   7.41   18.40 19.05           Endoderm Pharyngeal
## 18   6.89  3.66   6.68    4.11  3.28             Cephalic Mesoderm
## 19  10.38 13.57  10.66   15.85 11.68           Diencephalon Ventral
## 21  10.03 13.24  10.41   14.65 11.89                  Neural Crest
## 22  10.03 13.24  10.41   14.65 11.89                      Midbrain
## 26   7.03  5.79   6.98   17.95 18.78 Endoderm Pancreatic/Intestinal
## 29   0.97  0.45   0.88    1.32  0.40               Prechordal Plate
## 31   6.69  5.45   6.65   17.42 17.99                     Pronephros
## 32   3.10  0.88   2.89    1.71  0.87            Notochord Anterior
## 33   6.69  5.45   6.65   17.42 17.99           Hematopoeitic (RBI)
## 34   6.55  4.52   5.78    5.59  4.44                 Adaxial Cells
## 38   0.44  0.46   0.40    0.40  0.42                 EVL/Periderm
## 40   0.62  0.55   0.49    0.53  0.64         Primordial Germ Cells
## 43  10.46 13.65  10.93   16.11 11.53                Hindbrain R5+6
## 45  14.52 20.92  16.29   21.45 14.85          Placode Epibranchial
## 46  14.48 20.88  16.24   16.82 14.89            Placode Trigeminal
## 47  10.46 13.65  10.93   16.11 11.53                 Hindbrain R4
## 48  14.48 20.97  16.24   21.57 14.89      Placode Adenohypophyseal
## 49  14.48 20.97  16.84   21.57 14.89                  Placode Lens
## 50  10.07 13.25  11.24   16.37 11.48                 Hindbrain R3
## 51  14.52 20.74  16.44   21.28 14.85                 Placode Otic
## 52   6.86  5.43   6.75   17.67 18.15           Hematopoeitic (ICM)
## 53  14.65 20.60  16.28   21.14 14.85             Placode Olfactory
```

Some lineages are clearly more sensitive than others; those that branched early (such as the axial mesoderm – the notochord and prechordal plate) are very robust, even against this parameter, while those that branch later (such as the individual placodes, which are just beginning to form) are much more sensitive.

Modifying diffuson map parameters

We also built the tree using two additional diffusion map sigmas (7 & 9) that performed reasonably well (see "URD: Choosing Parameters - Diffusion Map Sigma"), while holding the biased random walk parameters constant.

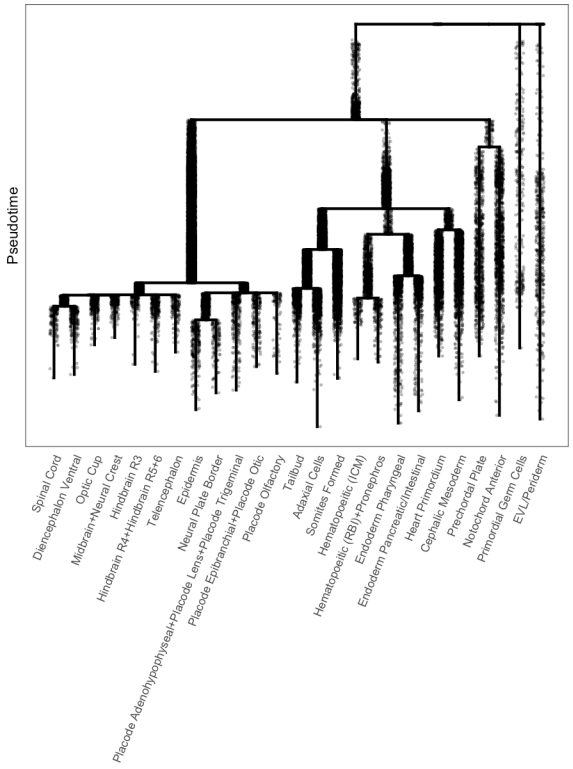
Compare Resultant Tree Structures

Much of the tree structure is preserved across all of these trees, though there is some variability.

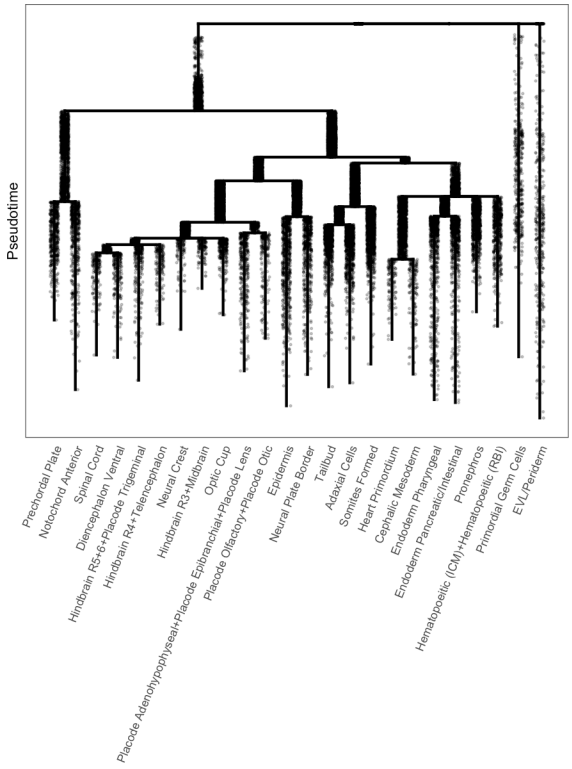
Sigma 7 resulted in a later separatation of the non-axial mesendoderm from the ectoderm, but otherwise produced a largely similar structure. Sigma 9 had more dramatic effects and produces the only tree so far that really violates known embryology – a portion of the endoderm ends up assigned in the neural ectoderm, while the optic cup ends up assigned in the mesendoderm. This illustrates that URD's performance is much more sensitive to the initial diffusion map that it operates on, and suggests that using the smallest possible sigma that does not cause disconnections is likely to work best.

```
sigma.tree.plots <- list(plotTree(object, title = "Sigma 8: Tree constructed in manuscript"),
    plotTree(object.dm7, title = "Sigma 7"), plotTree(object.dm9, title = "Sigma 9"))
grid.arrange(grobs = sigma.tree.plots, ncol = 2)
```

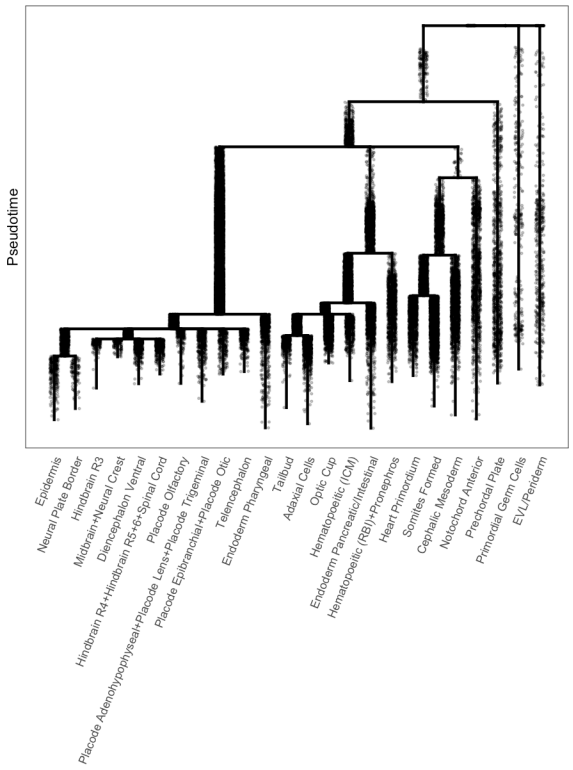
Sigma 8: Tree constructed in manuscript



Sigma 7



Sigma 9



Compare changes in trajectory membership across entire tree

We quantified the percentage of the assignments (of cell to trajectory) that changed in each tree. Each cell was determined as "in" or "not-in" each trajectory, and the percentage of those assignments that changed for each alternative tree relative to the tree presented in the paper was determined.

```
alts <- c("object.dm7", "object.dm9")

tree.change <- lapply(alts, function(alt) {

    object.alt <- get(alt)

    # Some tips combined immediately, but this could be different between trees.
    # Create a data frame for each original tip to the terminal segment that it
    # belongs to in each resultant tree.

    tree.terminal.segs <- data.frame(original.tip = object@tree$tips, row.names = object@tree$tips,
        stringsAsFactors = F)

    tree.terminal.segs[, 2:3] <- t(apply(tree.terminal.segs, 1, function(ot) {
        ot <- ot[1]
        if (ot %in% segTerminal(object)) {
            seg.1 <- ot
        } else {
            t <- ot
            while (!(t %in% segTerminal(object))) {
                t <- object@tree$segment.joins.initial[c(which(object@tree$segment.joins.initial$child.1 ==
                  t), which(object@tree$segment.joins.initial$child.2 == t)),
                  "parent"]
            }
            seg.1 <- t
        }
        if (ot %in% segTerminal(object.alt)) {
            seg.2 <- ot
        } else {
            t <- ot
            while (!(t %in% segTerminal(object.alt))) {
                t <- object.alt@tree$segment.joins.initial[c(which(object.alt@tree$segment.joins.initial$child.1 ==
                  t), which(object.alt@tree$segment.joins.initial$child.2 ==
                  t)), "parent"]
            }
            seg.2 <- t
        }
        return(c(seg.1, seg.2))
    }))

    # Now, get the cells in the tree overall for both trees so you only consider
    # those.
    cells.in.tree.original <- unique(unlist(object@tree$cells.in.segment))
    cells.in.tree.alt <- unique(unlist(object.alt@tree$cells.in.segment))
    cells.in.trees <- intersect(cells.in.tree.original, cells.in.tree.alt)

    # Now, for each lineage, figure out the number of cells in that lineage for
    # each tree
    tree.terminal.segs[, c(4:7)] <- t(apply(tree.terminal.segs, 1, function(ts) {
        cells.orig <- intersect(cellsAlongLineage(object, segments = ts[2],
            remove.root = F), cells.in.trees)
        cells.alt <- intersect(cellsAlongLineage(object.alt, segments = ts[3],
            remove.root = F), cells.in.trees)
        cells.both <- length(intersect(cells.orig, cells.alt))
        cells.orig.only <- length(setdiff(cells.orig, cells.alt))
        cells.alt.only <- length(setdiff(cells.alt, cells.orig))
        cells.neither <- length(cells.in.trees) - (cells.both + cells.orig.only +
            cells.alt.only)
        return(c(cells.both, cells.orig.only, cells.alt.only, cells.neither))
    }))

    names(tree.terminal.segs) <- c("tip.run", "tip.orig", "tip.alt", "cells.both",
        "cells.orig", "cells.alt", "cells.neither")

    return(tree.terminal.segs)
```

```
})
names(tree.change) <- alts
# Figure out how much each changed.

tree.changed.overall <- unlist(lapply(tree.change, function(tc) {
    tcsum <- apply(tc[, 4:7], 2, sum)
    changed <- round(((tcsum[2] + tcsum[3])/sum(tcsum)) * 100, digits = 2)
    return(changed)
}))

tree.changed.overall.present <- paste0(tree.changed.overall, "%")
names(tree.changed.overall.present) <- c("Sigma7", "Sigma9")

tree.changed.overall.present
```

```
##   Sigma7   Sigma9
##  "13.5%" "12.71%"
```

Again, the diffusion map parameter has a stonger effect than the biased walk parameters, though still most cells are robustly assigned to the same segments.

Compare changes in trajectory membership for each trajectory

We also wondered if some parts of the tree were more sensitive to others than the parameters of the reconstruction. So, we quantified the percentage change in each lineage, given the different parameters.

```
tree.changed.by.lineage <- as.data.frame(lapply(tree.change, function(tc) {
    apply(tc[, 4:7], 1, function(tcr) {
        changed <- round((tcr[2] + tcr[3])/sum(tcr[1:4]) * 100, digits = 2)
    })
}))

tip.names <- unique(object@group.ids[, c("ZF6S-Cluster-Num", "ZF6S-Cluster")])
tip.names <- tip.names[complete.cases(tip.names), ]
rownames(tip.names) <- tip.names[, "ZF6S-Cluster-Num"]
tree.changed.by.lineage$population <- tip.names[rownames(tree.changed.by.lineage),
    "ZF6S-Cluster"]

names(tree.changed.by.lineage) <- c("Sigma7", "Sigma9", "Population")

tree.changed.by.lineage
```

```
##    Sigma7 Sigma9                    Population
## 1   20.34  12.96                     Epidermis
## 2   13.40  43.37                     Optic Cup
## 3   14.62   9.71                       Tailbud
## 5   15.83   7.90                Heart Primordium
## 8   13.90  15.09                     Spinal Cord
## 10  20.70  12.95             Neural Plate Border
## 12  15.03  10.72                 Somites Formed
## 16  14.15  14.81                  Telencephalon
## 17  15.64  35.28             Endoderm Pharyngeal
## 18  14.92   5.56               Cephalic Mesoderm
## 19  13.88  14.66           Diencephalon Ventral
## 21  13.76  14.61                   Neural Crest
## 22  13.94  14.61                       Midbrain
## 26  15.61  13.35 Endoderm Pancreatic/Intestinal
## 29   3.67   1.37                Prechordal Plate
## 31  16.02   7.59                      Pronephros
## 32   3.35   1.75              Notochord Anterior
## 33  15.43   7.59             Hematopoeitic (RBI)
## 34  15.59  10.41                   Adaxial Cells
## 38   0.51   0.52                   EVL/Periderm
## 40   1.05   0.54           Primordial Germ Cells
## 43  14.50  15.15                  Hindbrain R5+6
## 45  14.30  13.42            Placode Epibranchial
## 46  17.99  13.52              Placode Trigeminal
## 47  14.45  15.15                   Hindbrain R4
## 48  14.36  13.52        Placode Adenohypophyseal
## 49  14.36  13.52                   Placode Lens
## 50  13.86  14.64                   Hindbrain R3
```

```
## 51  14.23  13.42                  Placode Otic
## 52  15.14  12.92           Hematopoeitic (ICM)
## 53  14.01  13.49             Placode Olfactory
```

Again, the choice of diffusion map makes a larger difference. Here, changes seem to be distributed relatively equally across cell types, though some of the earliest branching types seem more robust (such as the notochord, prechordal plate, EVL, and primordial germ cells). For sigma 9, as expected, there are dramatic changes in the assignment to the optic cup and pharyngeal endoderm – the two populations that end up misconnected in the tree.

Connect modules between stages

```
suppressWarnings(library("knitr"))
suppressWarnings(library("gplots"))
suppressWarnings(library("igraph"))
opts_chunk$set(tidy.opts=list(width.cutoff=80),tidy=TRUE,dev="png",dpi=150)
```

Load the NMF results for each of the 12 stages

A best K (number of modules or `n_component` argument used for running NMF) is picked for each stage based on the stability of the results from 10 NMF runs with random initial conditions. The results are then organized into two lists: one contains all the matrices Cs (modules by cells), and one for all the matrices Gs (genes by modules).

```
load_obj <- function(file.path) {
    temp.space <- new.env()
    obj <- load(file.path, temp.space)
    obj2 <- get(obj, temp.space)
    rm(temp.space)
    return(obj2)
}

DSHIGH_k = c(10)
DSOBLONG_k = c(11)
DSDOME_k = c(17)
DS30_k = c(15)
DS50_k = c(20)
DSS_k = c(25)
DS60_k = c(25)
DS75_k = c(24)
DS90_k = c(45)
DSB_k = c(40)
DS3S_k = c(31)
DS6S_k = c(42)

stages = c("HIGH", "OBLONG", "DOME", "30", "50", "S", "60", "75", "90", "B", "3S",
    "6S")

NMF_list = list()
for (stage in stages) {
    stage_k = get(paste0("DS", stage, "_k"))[1]
    NMF_obj = load_obj(paste0("./DS_ZF", stage, "/result_tbls.Robj"))
    NMF_list[[paste0("DS", stage)]] = NMF_obj[[paste0("K=", stage_k)]][["rep0"]]
}

DS_C <- list()
DS_G <- list()

ds_genes = c()
for (stage in stages) {
    DS_C[[stage]] <- NMF_list[[paste0("DS", stage)]][["C"]]
    DS_G[[stage]] <- NMF_list[[paste0("DS", stage)]][["G"]]
    colnames(DS_G[[stage]]) = rownames(DS_C[[stage]])
    ds_genes = c(ds_genes, rownames(DS_G[[stage]]))
}
```

Find and remove modules that are primarily driven by batch and noise from each stage

Batch modules are found using the `BatchGene` function in *Seurat* package. Noise modules are defined as the ones that are primarily driven by a single gene (the top ranked gene has a weight more than 3 times the weight of the second ranked gene). Matrices G and C with the batch and noise modules removed were again saved in two lists.

```
library("Seurat")
```

```
## Loading required package: ggplot2
## Loading required package: cowplot
## Warning: package 'cowplot' was built under R version 3.4.3
##
## Attaching package: 'cowplot'
## The following object is masked from 'package:ggplot2':
##
##     ggsave
```

```
DS_C_use <- list()
DS_G_use <- list()

maxScl <- function(df, dir = "row", max_value = NULL, log_space = TRUE) {
    if (dir == "row") {
        dir = 1
    } else if (dir == "col") {
        dir = 2
    } else {
        print("dir must be 'row' or 'col'.")
        return
    }
    if (is.null(max_value)) {
        max_value = median(apply(df, dir, max))
    }
    if (log_space) {
        df = expm1(df)
        max_value = expm1(max_value)
    }
    df_scl = sweep(df, dir, apply(df, dir, max), "/")
    df_scl = df_scl * max_value
    if (log_space) {
        df_scl = log1p(df_scl)
    }
    return(df_scl)
}

rmByCell <- function(scData, low = 1) {
    bData = scData > 0
    # sum up each row in the binary matrix for cell numbers
    num.cell = apply(bData, 1, sum)
    rm.ind = which(num.cell <= low)
    scData.f = scData
    # print(paste('removing',length(rm.ind),'genes...'))
    if (length(rm.ind) > 0) {
        scData.f = scData[-rm.ind, ]
    }
    # now there could be cells with no gene detection. remove them
    rmByGenes(scData.f, lmt = 0)
    return(scData.f)
}

rmByGenes <- function(scData, lmt) {
    # first creat a binary matrix for gene detection
    cptr = scData > 0
    # then sum up each column in the binary matrix for gene numbers
    num.cptr = apply(cptr, 2, sum)
    rm.ind = which(num.cptr <= lmt)
    scData.f = scData
    if (length(rm.ind) > 0) {
        # print(paste('removing',length(rm.ind),'cells with fewer than',lmt,'genes...'))
        scData.f = scData[, -rm.ind]
    }
    # now there could be genes with no detection in any cells. remove them
    cptr = scData.f > 0
    num.cell = apply(cptr, 1, sum)
    rm.ind = which(num.cell == 0)
    if (length(rm.ind) > 0) {
        scData.f = scData.f[-rm.ind, ]
    }
    return(scData.f)
}
```

```
for (stage in stages) {
    ZF_seurat = new("seurat", raw.data = DS_C[[stage]])
    ZF_seurat = Setup(ZF_seurat, project = "ds", min.cells = 2, names.field = 3,
        names.delim = "_", do.logNormalize = F, is.expr = 0.01, min.genes = 1)
    cut_off = 0.73
    if (stage %in% c("B")) {
        cut_off = 0.75
    }
    batch_module = BatchGene(ZF_seurat, idents.use = levels(ZF_seurat@ident), genes.use = rownames(ZF_seurat@data),
        auc.cutoff = cut_off)
    print(paste("Stage:", stage))
    print(paste("number of batches:", length(levels(ZF_seurat@ident))))
    print("Batch modules:")
    print(batch_module)
    weigh_st = apply(DS_G[[stage]], 2, sort)
    weigh_rat = weigh_st[dim(weigh_st)[1], ]/weigh_st[dim(weigh_st)[1] - 1, ]
    nois = weigh_rat[which(weigh_rat > 3)]
    if (length(nois) > 0) {
        print("Noise modules:")
        print(names(nois))
    }
    batch_module = union(batch_module, names(nois))
    # print(batch_module)
    DS_C_use[[stage]] <- DS_C[[stage]][setdiff(rownames(DS_C[[stage]]), batch_module),
        ]
    DS_C_use[[stage]] <- maxScl(DS_C_use[[stage]], log_space = F)
    DS_G_use[[stage]] <- DS_G[[stage]][, setdiff(colnames(DS_G[[stage]]), batch_module)]
    DS_G_use[[stage]] <- rmByCell(DS_G_use[[stage]], low = 0)
    DS_G_use[[stage]] <- maxScl(DS_G_use[[stage]], dir = "col", log_space = F)
}
```

```
## [1] "Stage: HIGH"
## [1] "number of batches: 2"
## [1] "Batch modules:"
## [1] "8" "5" "0" "1"
## [1] "Stage: OBLONG"
## [1] "number of batches: 2"
## [1] "Batch modules:"
## [1] "2"
## [1] "Stage: DOME"
## [1] "number of batches: 1"
## [1] "Batch modules:"
## NULL
## [1] "Stage: 30"
## [1] "number of batches: 2"
## [1] "Batch modules:"
## [1] "3" "5"
## [1] "Stage: 50"
## [1] "number of batches: 4"
## [1] "Batch modules:"
## [1] "3"  "4"  "10" "16" "9"  "7"  "0"
## [1] "Stage: S"
## [1] "number of batches: 1"
## [1] "Batch modules:"
## NULL
## [1] "Stage: 60"
## [1] "number of batches: 3"
## [1] "Batch modules:"
## [1] "5"  "15" "13" "17"
## [1] "Noise modules:"
## [1] "23"
## [1] "Stage: 75"
## [1] "number of batches: 3"
## [1] "Batch modules:"
## [1] "4"  "23" "21" "16"
## [1] "Noise modules:"
## [1] "0"
## [1] "Stage: 90"
## [1] "number of batches: 3"
## [1] "Batch modules:"
## [1] "24" "9"  "23"
```

```
## [1] "Noise modules:"
##  [1] "6"  "29" "31" "33" "34" "35" "36" "37" "38" "39" "40" "41" "43" "44"
## [1] "Stage: B"
## [1] "number of batches: 4"
## [1] "Batch modules:"
## [1] "15"
## [1] "Noise modules:"
##  [1] "20" "26" "27" "28" "30" "32" "34" "35" "36" "37" "38" "39"
## [1] "Stage: 3S"
## [1] "number of batches: 1"
## [1] "Batch modules:"
## NULL
## [1] "Noise modules:"
## [1] "28" "29"
## [1] "Stage: 6S"
## [1] "number of batches: 2"
## [1] "Batch modules:"
## [1] "25" "8"
## [1] "Noise modules:"
##  [1] "24" "28" "30" "31" "32" "33" "36" "38" "39" "41"
```

Print out the size of matrix G at each stage (we will use these matrices to build the tree of connected modules)

```
for (stage in stages) {
    print(stage)
    # print(dim(DS_C_use[[stage]]))
    print(dim(DS_G_use[[stage]]))
}
```

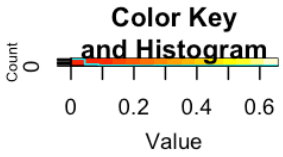
```
## [1] "HIGH"
## [1] 1262    6
## [1] "OBLONG"
## [1] 1211   10
## [1] "DOME"
## [1] 1583   17
## [1] "30"
## [1] 1553   13
## [1] "50"
## [1] 1721   13
## [1] "S"
## [1] 1573   25
## [1] "60"
## [1] 1749   20
## [1] "75"
## [1] 1809   19
## [1] "90"
## [1] 1833   28
## [1] "B"
## [1] 1856   27
## [1] "3S"
## [1] 1825   29
## [1] "6S"
## [1] 1854   30
```

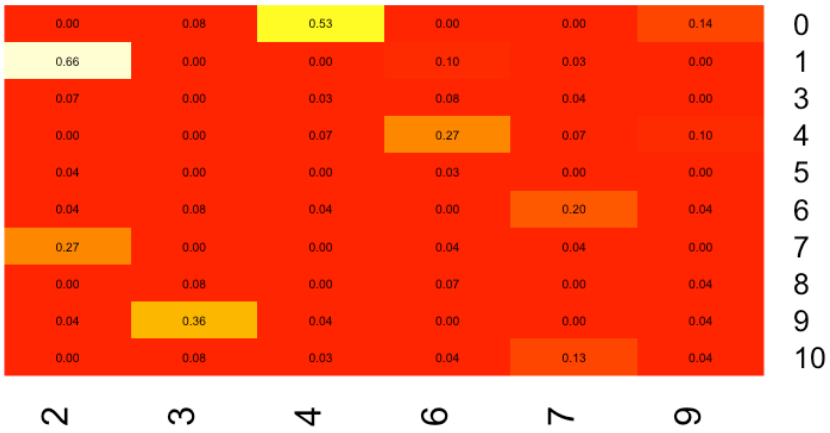
Calculate the weighted overlap between pairs of gene modules in adjacent stages

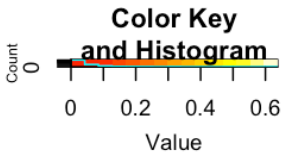
Only the top 25 genes in each module were used in this calculation (see methods in the paper). The results of the overlap scores are visualized in heat maps.

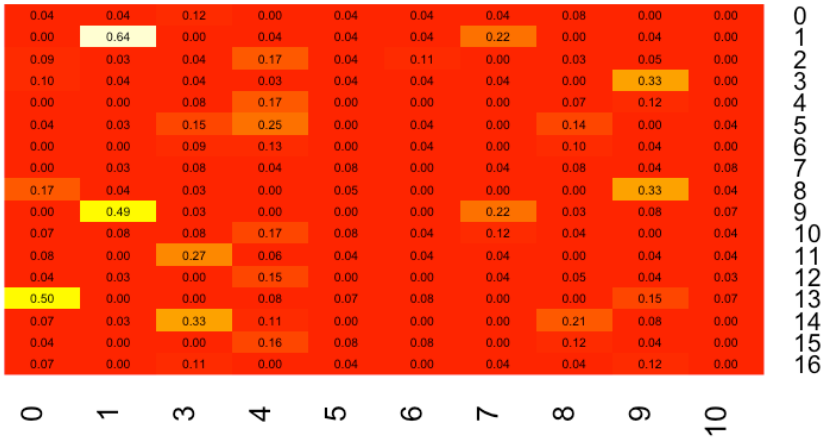
```
Weigh_intersect <- function(M.ind, Data1, Data2, numGene) {
    i = M.ind[1]
    j = M.ind[2]
    Data1M = Data1[, i, drop = F]
    Data2M = Data2[, j, drop = F]
    topGenes1 = rownames(Data1)[order(Data1M, decreasing = T)[1:numGene]]
    topGenes2 = rownames(Data2)[order(Data2M, decreasing = T)[1:numGene]]
```

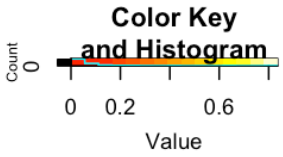
```
    inter_genes = intersect(topGenes1, topGenes2)
    weighted_inter = (sum(Data1M[inter_genes, ]) + sum(Data2M[inter_genes, ]))/(sum(Data1M[topGenes1,
        ]) + sum(Data2M[topGenes2, ]))
    return(weighted_inter)
}
Calc_intersect <- function(Data1, Data2, num_top = 25, weigh = F) {
    Data1 = sweep(Data1, 2, apply(Data1, 2, max), "/")
    Data2 = sweep(Data2, 2, apply(Data2, 2, max), "/")

    genes.com = intersect(rownames(Data1), rownames(Data2))
    Data1 = Data1[genes.com, ]
    Data2 = Data2[genes.com, ]
    num.spl1 = dim(Data1)[2]
    num.spl2 = dim(Data2)[2]
    cor.M = matrix(0, nrow = num.spl2, ncol = num.spl1)
    num.ind = num.spl1 * num.spl2
    M.ind = vector("list", length = num.ind)
    k = 1
    for (i in 1:num.spl1) {
        for (j in 1:num.spl2) {
            M.ind[[k]] = c(i, j)
            k = k + 1
        }
    }

    if (weigh) {
        cor.M.vec = lapply(1:num.ind, function(x) Weigh_intersect(M.ind[[x]], Data1,
            Data2, num_top))
    } else {
        cor.M.vec = lapply(1:num.ind, function(x) length(intersect(rownames(Data1)[order(Data1[,
            M.ind[[x]][1]], decreasing = T)[1:num_top]], rownames(Data2)[order(Data2[,
            M.ind[[x]][2]], decreasing = T)[1:num_top]]))/num_top)
    }

    for (i in 1:num.ind) {
        ind1 = M.ind[[i]][1]
        ind2 = M.ind[[i]][2]
        cor.M[ind2, ind1] = unlist(cor.M.vec[i])
    }
    corDF = data.frame(cor.M, row.names = colnames(Data2))
    colnames(corDF) = colnames(Data1)
    return(corDF)
}

G_int <- list()
for (i in 1:(length(stages) - 1)) {
    stage = stages[i]
    stage_next = stages[i + 1]
    gene_use = intersect(rownames(DS_G_use[[stage]]), rownames(DS_G_use[[stage_next]]))
    G_stage = DS_G_use[[stage]][gene_use, ]
    G_stage_next = DS_G_use[[stage_next]][gene_use, ]
    num_module = dim(G_stage)[2]
    num_module_next = dim(G_stage_next)[[2]]
    G_int[[stage]] <- Calc_intersect(G_stage, G_stage_next, num_top = 25, weigh = T)
    ## returns overlap scores in a matrix, colnames are modules at this stage,
    ## rownames are modules at next stage
    xval <- formatC(as.matrix(G_int[[stage]]), format = "f", digits = 2)
    heatmap.2(as.matrix(G_int[[stage]]), Rowv = FALSE, Colv = FALSE, dendrogram = "none",
        xlab = stage, ylab = stage_next, trace = "none", cellnote = xval, notecol = "black",
        notecex = 0.5)
}
```

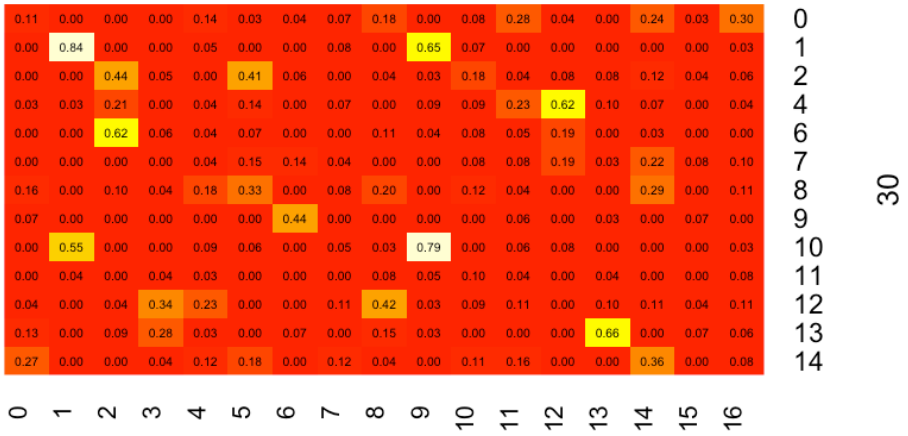

Color Key
and Histogram
Count
0
0
0.2
0.4
0.6
Value


0.00 0.08 0.53 0.00 0.00 0.14
0.66 0.00 0.00 0.10 0.03 0.00
0.07 0.00 0.03 0.08 0.04 0.00
0.00 0.00 0.07 0.27 0.07 0.10
0.04 0.00 0.00 0.03 0.00 0.00
0.04 0.08 0.04 0.00 0.20 0.04
0.27 0.00 0.00 0.04 0.04 0.00
0.00 0.08 0.00 0.07 0.00 0.04
0.04 0.36 0.04 0.00 0.00 0.04
0.00 0.08 0.03 0.04 0.13 0.04
0
1
3
4
5
6
7
8
9
10
OBLONG
2
3
4
6
7
9
HIGH

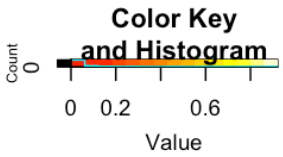

Color Key
and Histogram
Count
0
0
0.2
0.4
0.6
Value

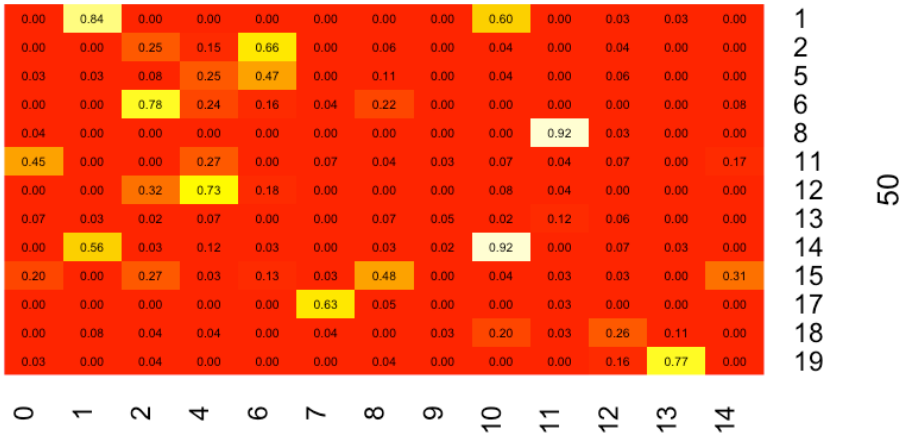

0.04 0.04 0.12 0.00 0.04 0.04 0.04 0.08 0.00 0.00
0.00 0.64 0.00 0.04 0.04 0.04 0.22 0.00 0.04 0.00
0.09 0.03 0.04 0.17 0.04 0.11 0.00 0.03 0.05 0.00
0.10 0.04 0.04 0.03 0.04 0.04 0.04 0.00 0.33 0.00
0.00 0.00 0.08 0.17 0.00 0.00 0.00 0.07 0.12 0.00
0.04 0.03 0.15 0.25 0.00 0.04 0.00 0.14 0.00 0.04
0.00 0.00 0.09 0.13 0.00 0.04 0.00 0.10 0.04 0.00
0.00 0.03 0.08 0.04 0.08 0.00 0.04 0.08 0.04 0.08
0.17 0.04 0.03 0.00 0.05 0.00 0.00 0.00 0.33 0.04
0.00 0.49 0.03 0.00 0.00 0.00 0.22 0.03 0.08 0.07
0.07 0.08 0.08 0.17 0.08 0.04 0.12 0.04 0.00 0.04
0.08 0.00 0.27 0.06 0.04 0.04 0.04 0.00 0.04 0.04
0.04 0.03 0.00 0.15 0.00 0.00 0.04 0.05 0.04 0.03
0.50 0.00 0.00 0.08 0.07 0.08 0.00 0.00 0.15 0.07
0.07 0.03 0.33 0.11 0.00 0.00 0.00 0.21 0.08 0.00
0.04 0.00 0.00 0.16 0.08 0.08 0.00 0.12 0.04 0.00
0.07 0.00 0.11 0.00 0.04 0.00 0.04 0.04 0.12 0.00
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
DOME
0
1
3
4
5
6
7
8
9
10
OBLONG


Color Key
and Histogram
Count
0
0
0.2
0.6
Value

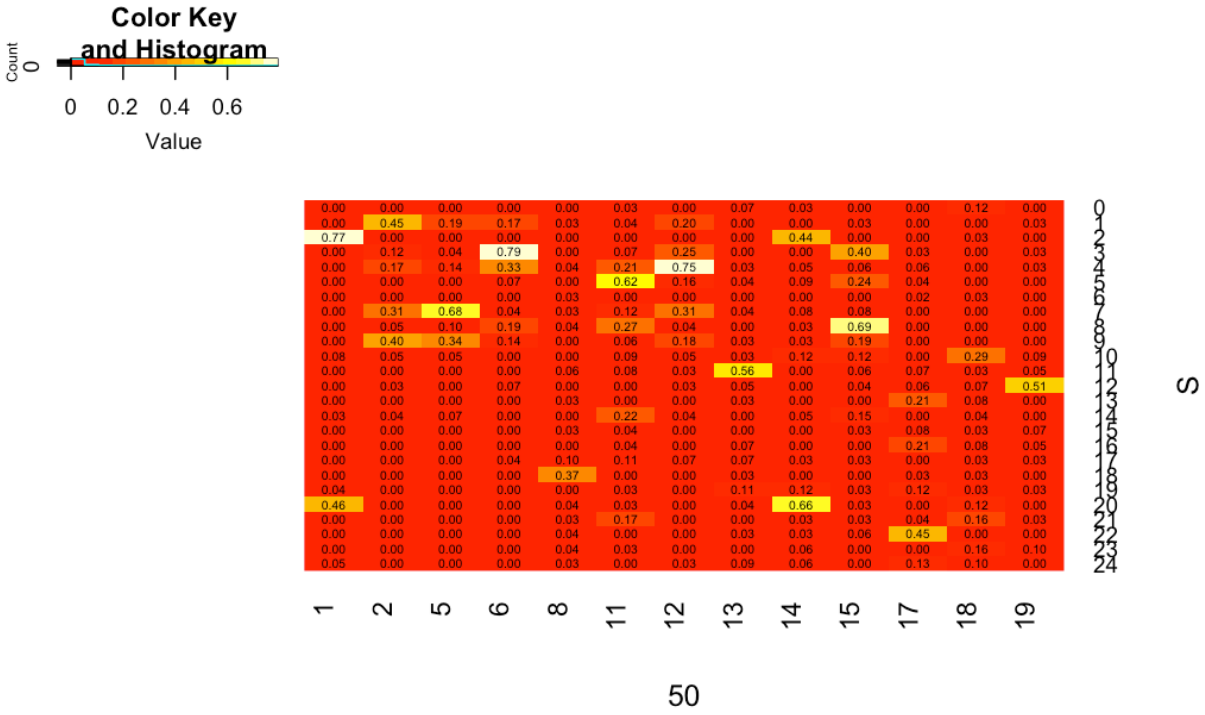

0.11 0.00 0.00 0.00 0.14 0.03 0.04 0.07 0.18 0.00 0.08 0.28 0.04 0.00 0.24 0.03 0.30
0.00 0.84 0.00 0.00 0.05 0.00 0.00 0.08 0.00 0.65 0.07 0.00 0.00 0.00 0.00 0.00 0.03
0.00 0.00 0.44 0.05 0.00 0.41 0.06 0.00 0.04 0.03 0.18 0.04 0.08 0.08 0.12 0.04 0.06
0.03 0.03 0.21 0.00 0.04 0.14 0.00 0.07 0.00 0.09 0.09 0.23 0.62 0.10 0.07 0.00 0.04
0.00 0.00 0.62 0.06 0.04 0.07 0.00 0.00 0.11 0.04 0.08 0.05 0.19 0.00 0.03 0.00 0.00
0.00 0.00 0.00 0.00 0.04 0.15 0.14 0.04 0.00 0.00 0.08 0.08 0.19 0.03 0.22 0.08 0.10
0.16 0.00 0.10 0.04 0.18 0.33 0.00 0.08 0.20 0.00 0.12 0.04 0.00 0.00 0.29 0.00 0.11
0.07 0.00 0.00 0.00 0.00 0.00 0.44 0.00 0.00 0.00 0.00 0.06 0.00 0.03 0.00 0.07 0.00
0.00 0.55 0.00 0.00 0.09 0.06 0.00 0.05 0.03 0.79 0.00 0.06 0.08 0.00 0.00 0.00 0.03
0.00 0.04 0.00 0.04 0.03 0.00 0.00 0.00 0.08 0.05 0.10 0.04 0.00 0.04 0.00 0.00 0.08
0.04 0.00 0.04 0.34 0.23 0.00 0.00 0.11 0.42 0.03 0.09 0.11 0.00 0.10 0.11 0.04 0.11
0.13 0.00 0.09 0.28 0.03 0.00 0.07 0.00 0.15 0.03 0.00 0.00 0.00 0.66 0.00 0.07 0.06
0.27 0.00 0.00 0.04 0.12 0.18 0.00 0.12 0.04 0.00 0.11 0.16 0.00 0.00 0.36 0.00 0.08
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
0 1 2 4 6 7 8 9 10 11 12 13 14
30

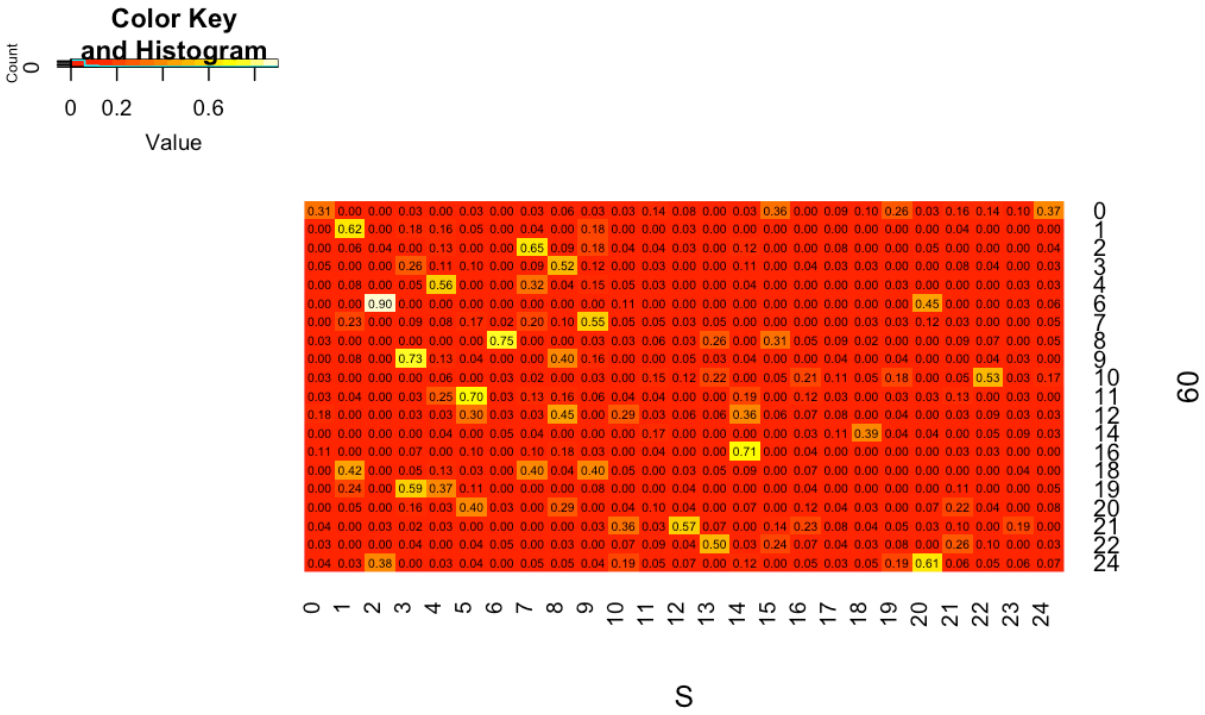
DOME

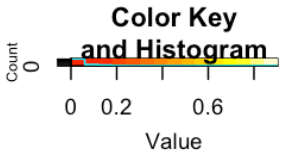

Color Key
and Histogram
Count
0
0
0.2
0.6
Value

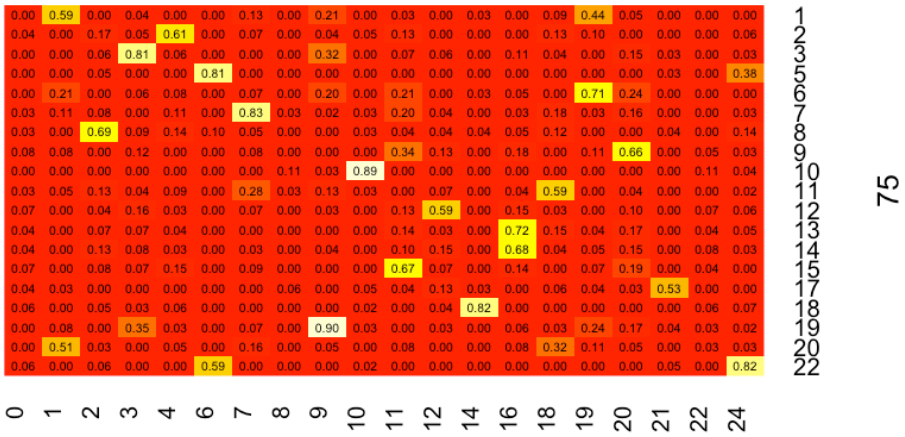

0.00 0.84 0.00 0.00 0.00 0.00 0.00 0.00 0.60 0.00 0.03 0.03 0.00
0.00 0.00 0.25 0.15 0.66 0.00 0.06 0.00 0.04 0.00 0.04 0.00 0.00
0.03 0.03 0.08 0.25 0.47 0.00 0.11 0.00 0.04 0.00 0.06 0.00 0.00
0.00 0.00 0.78 0.24 0.16 0.04 0.22 0.00 0.00 0.00 0.00 0.00 0.08
0.04 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.92 0.03 0.00 0.00
0.45 0.00 0.00 0.27 0.00 0.07 0.04 0.03 0.07 0.04 0.07 0.00 0.17
0.00 0.00 0.32 0.73 0.18 0.00 0.00 0.00 0.08 0.04 0.00 0.00 0.00
0.07 0.03 0.02 0.07 0.00 0.00 0.07 0.05 0.02 0.12 0.06 0.00 0.00
0.00 0.56 0.03 0.12 0.03 0.00 0.03 0.02 0.92 0.00 0.07 0.03 0.00
0.20 0.00 0.27 0.03 0.13 0.03 0.48 0.00 0.04 0.03 0.03 0.00 0.31
0.00 0.00 0.00 0.00 0.00 0.63 0.05 0.00 0.00 0.03 0.00 0.00 0.00
0.00 0.08 0.04 0.04 0.00 0.04 0.00 0.03 0.20 0.03 0.26 0.11 0.00
0.03 0.00 0.04 0.00 0.00 0.00 0.04 0.00 0.00 0.00 0.16 0.77 0.00
0 1 2 4 6 7 8 9 10 11 12 13 14
1 2 5 6 8 11 12 13 14 15 17 18 19
50

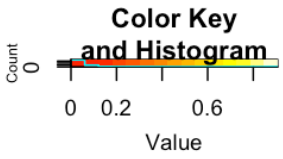
30

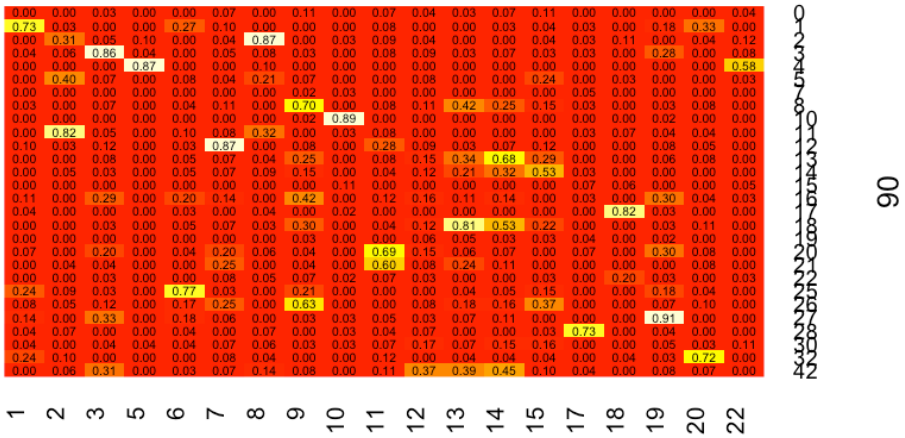

Color Key
and Histogram
Count
0
0 0.2 0.4 0.6
Value
1 2 5 6 8 11 12 13 14 15 17 18 19
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
S
50

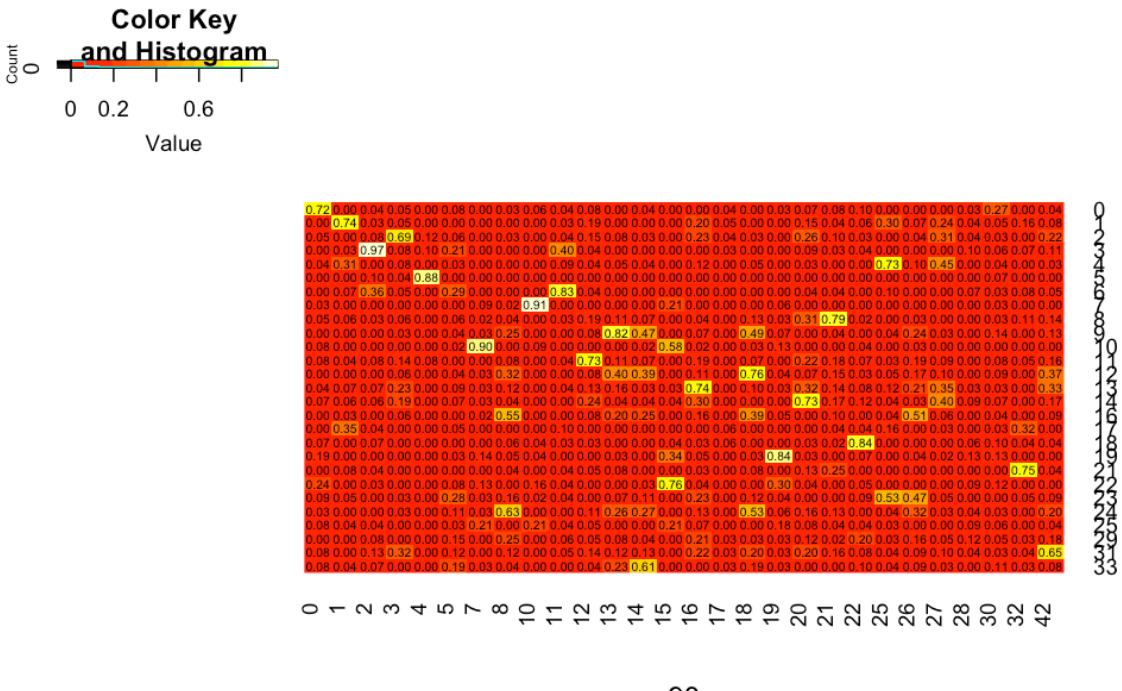

Color Key
and Histogram
Count
0
0 0.2 0.6
Value
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
S
60

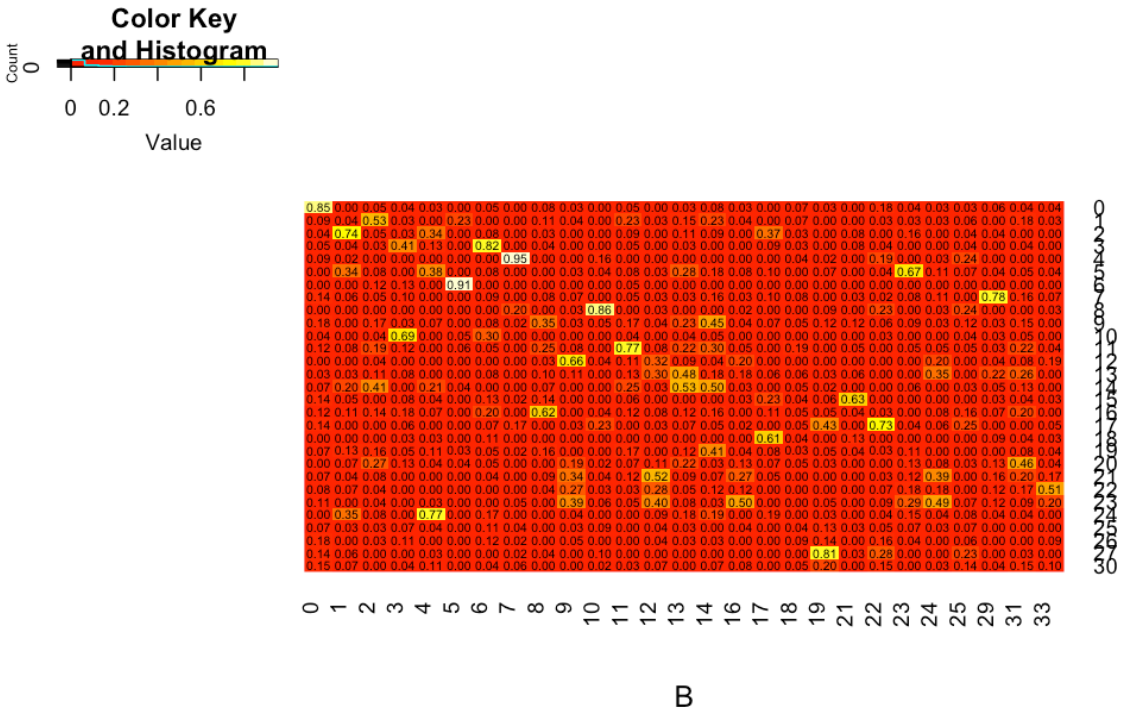

Color Key
and Histogram
Count
0
0
0.2
0.6
Value

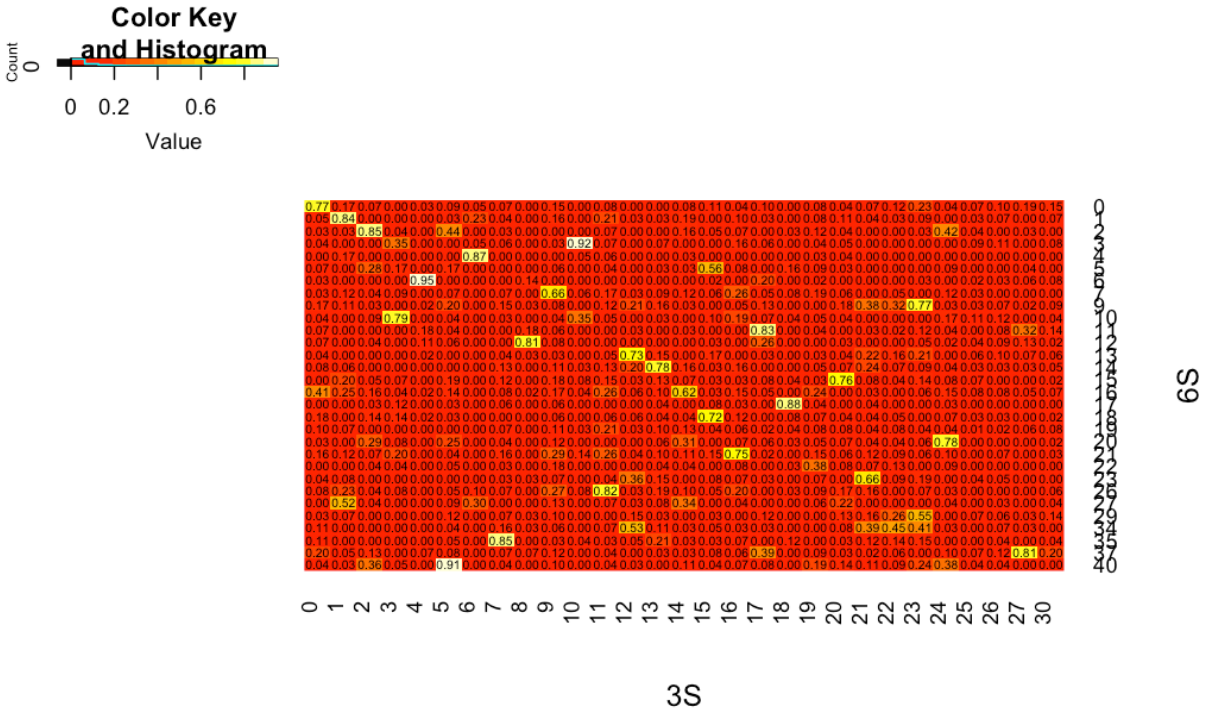

0 1 2 3 4 6 7 8 9 10 11 12 14 16 18 19 20 21 22 24
60
1 2 3 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 22
75


Color Key
and Histogram
Count
0
0
0.2
0.6
Value


1 2 3 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 22
75
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 25 26 27 28 32 42
90


Color Key
and Histogram
Count
0
0 0.2 0.6
Value
B
90


Color Key
and Histogram
Count
0
0 0.2 0.6
Value
3S
B



Filter out modules that have poor connection to modules in both adjacent stages

Modules that have <20% overlap with every module in the two adjacent stages are removed. Most of these modules are enriched with ubiquitously or lowly expressed genes.

```
mod_kp = list()
for (i in 1:length(stages)) {
    stage = stages[i]
    if (i > 1) {
        stage_pre = stages[i - 1]
    }
    if (i < length(stages)) {
        stage_next = stages[i + 1]
        G_cor_stage = G_int[[stage]]
        G_dim = dim(G_cor_stage)
        ## if a module has poor correlation with all modules in the next stage and
        ## previous stage, it is eliminated from the correlation matrix to reduce later
        G_cor_max = apply(G_cor_stage, 2, max)
        with_des = colnames(G_cor_stage)[which(G_cor_max > 0.2)]
        no_des = setdiff(colnames(G_cor_stage), with_des)
    }
    if (i > 1) {
        G_cor_stage_pre = G_int[[stage_pre]]
        G_cor_max_pre = apply(G_cor_stage_pre, 1, max)
        with_ans = rownames(G_cor_stage_pre)[which(G_cor_max_pre > 0.2)]
        no_ans = setdiff(rownames(G_cor_stage_pre), with_ans)
        if (i < length(stages)) {
            mod_kp[[stage]] = union(with_des, with_ans)
            mod_rm = intersect(no_des, no_ans)
        } else {
            mod_kp[[stage]] = with_ans
            mod_rm = no_ans
        }
    } else {
        mod_kp[[stage]] = with_des
        mod_rm = no_des
    }
    if (length(mod_rm) > 0) {
        print(stage)
        print(mod_rm)
```

```
    }
}
```

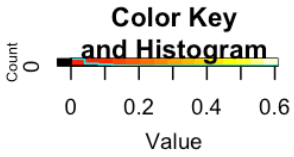
```
## [1] "HIGH"
## [1] "7" "9"
## [1] "OBLONG"
## [1] "5"  "6"  "10"
## [1] "DOME"
## [1] "7"  "10" "15"
## [1] "S"
## [1] "17" "23"
## [1] "3S"
## [1] "25" "26"
```

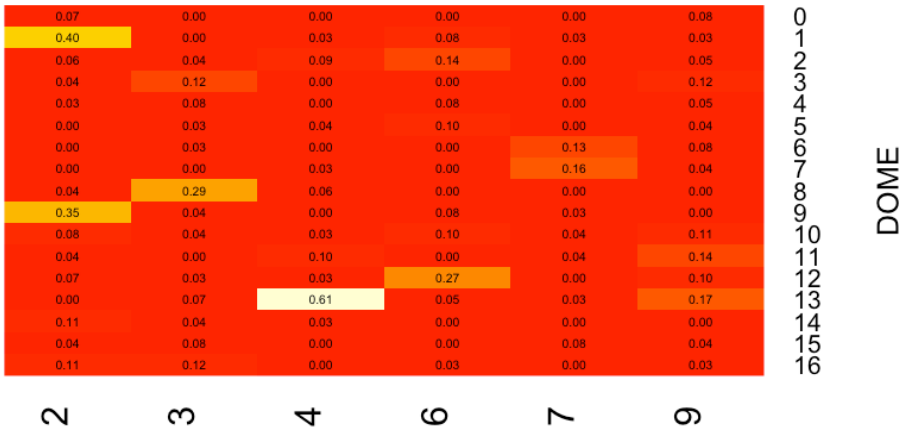
```
G_int_use <- list()
for (i in 1:(length(stages) - 1)) {
    stage = stages[i]
    stage_next = stages[i + 1]
    G_cor_stage = G_int[[stage]]
    G_int_use[[stage]] = G_cor_stage[mod_kp[[stage_next]], mod_kp[[stage]]]
}
```

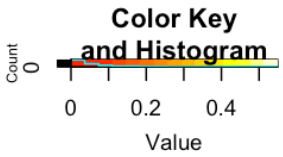
Calculate overlap between modules in every other stage

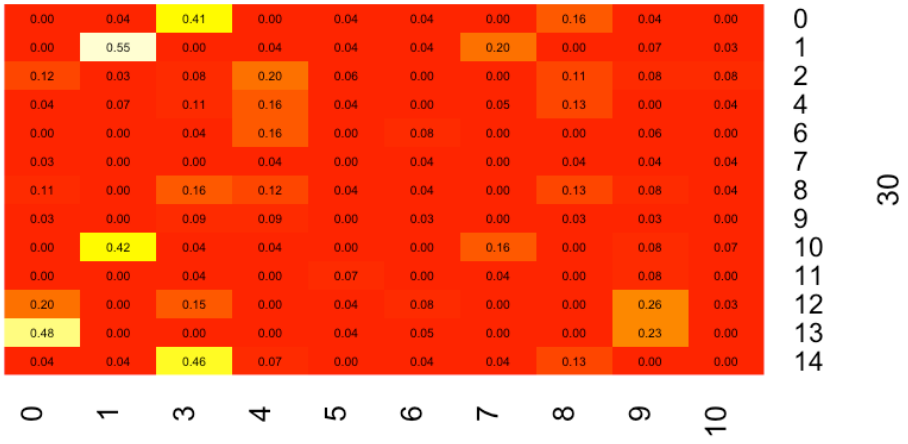
If a stage was not deeply or comprehensively sampled and sequenced, we might not be able to recover certain modules from that stage. This could potentially create dis-connections in the module lineages. In order to produce continuous module lineages when there is potential occasional drop-out of modules, we allow modules separated by one stage to connect to each other when connection to immediate neighbouring stage is not found.

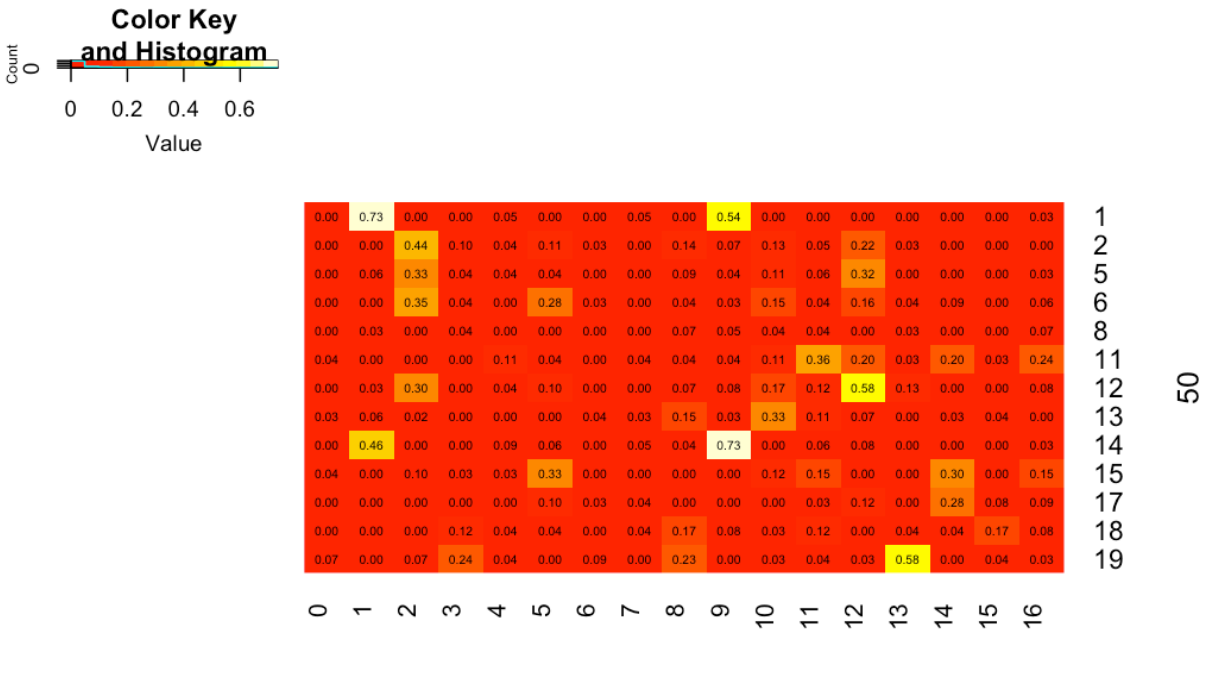
```
G_int2 <- list()
for (i in 1:(length(stages) - 2)) {
    stage = stages[i]
    stage_next = stages[i + 2]
    gene_use = intersect(rownames(DS_G_use[[stage]]), rownames(DS_G_use[[stage_next]]))
    G_stage = DS_G_use[[stage]][gene_use, ]
    G_stage_next = DS_G_use[[stage_next]][gene_use, ]
    num_module = dim(G_stage)[2]
    num_module_next = dim(G_stage_next)[[2]]
    G_int2[[stage]] <- Calc_intersect(G_stage, G_stage_next, num_top = 25, weigh = T)
    ## returns matrix of overlap scores, colnames are modules at this stage, rownames
    ## are modules at next stage
    xval <- formatC(as.matrix(G_int2[[stage]]), format = "f", digits = 2)
    heatmap.2(as.matrix(G_int2[[stage]]), Rowv = FALSE, Colv = FALSE, dendrogram = "none",
        xlab = stage, ylab = stage_next, trace = "none", cellnote = xval, notecol = "black",
        notecex = 0.5)
}
```

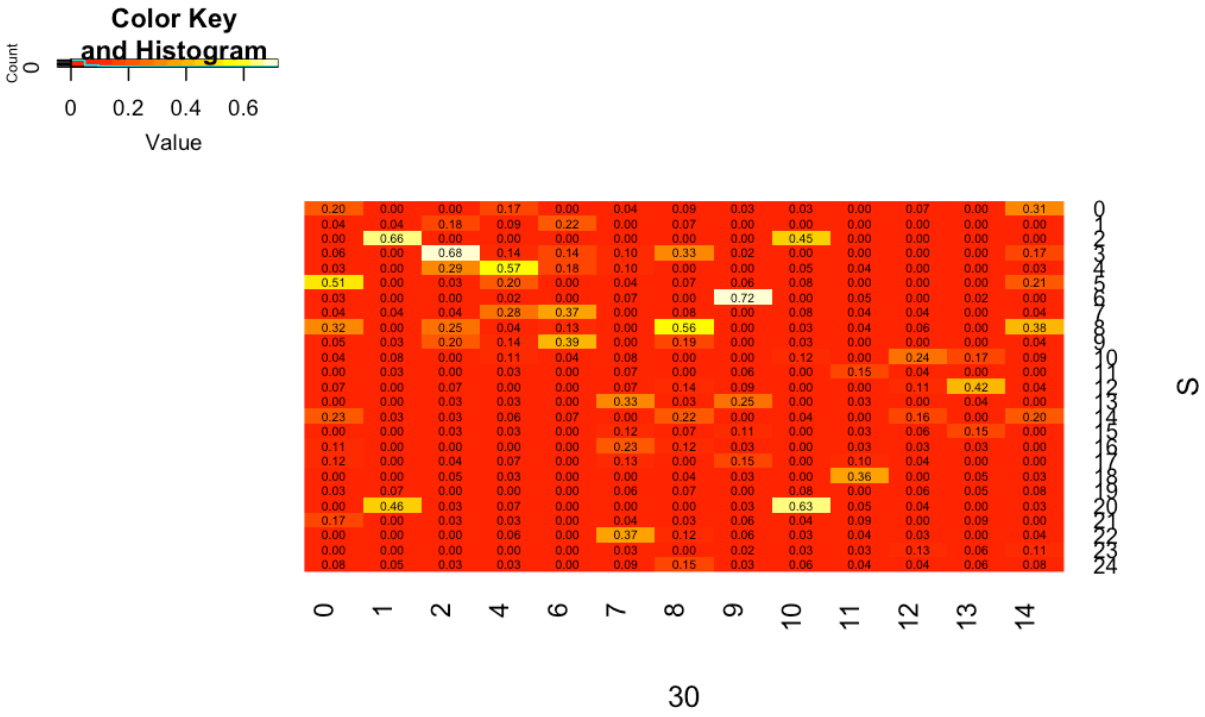

Color Key
and Histogram
Count
0
0
0.2
0.4
0.6
Value

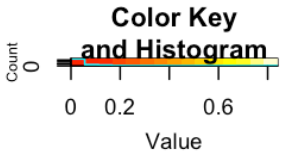

0.07 0.00 0.00 0.00 0.00 0.08
0.40 0.00 0.03 0.08 0.03 0.03
0.06 0.04 0.09 0.14 0.00 0.05
0.04 0.12 0.00 0.00 0.00 0.12
0.03 0.08 0.00 0.08 0.00 0.05
0.00 0.03 0.04 0.10 0.00 0.04
0.00 0.03 0.00 0.00 0.13 0.08
0.00 0.00 0.03 0.00 0.16 0.04
0.04 0.29 0.06 0.00 0.00 0.00
0.35 0.04 0.00 0.08 0.03 0.00
0.08 0.04 0.03 0.10 0.04 0.11
0.04 0.00 0.10 0.00 0.04 0.14
0.07 0.03 0.03 0.27 0.00 0.10
0.00 0.07 0.61 0.05 0.03 0.17
0.11 0.04 0.03 0.00 0.00 0.00
0.04 0.08 0.00 0.00 0.08 0.04
0.11 0.12 0.00 0.03 0.00 0.03
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
DOME
2
3
4
6
7
9
HIGH

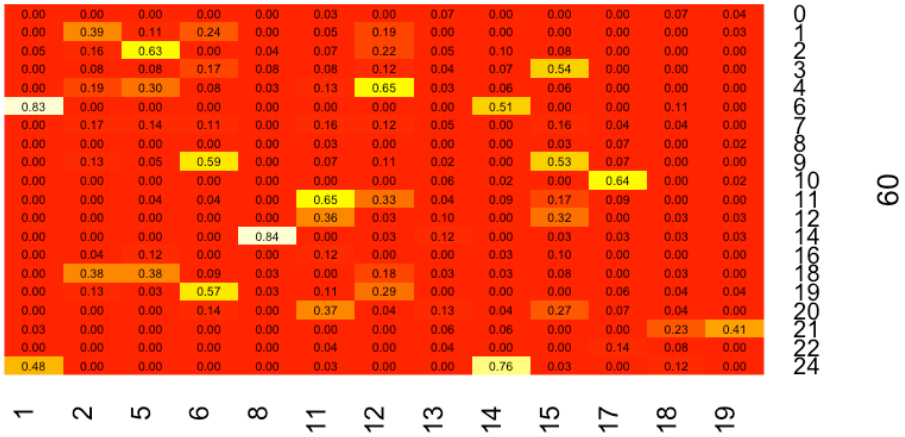

Color Key
and Histogram
Count
0
0
0.2
0.4
Value

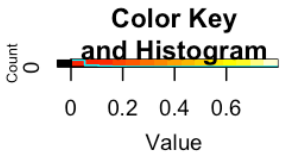

0.00 0.04 0.41 0.00 0.04 0.04 0.00 0.16 0.04 0.00
0.00 0.55 0.00 0.04 0.04 0.04 0.20 0.00 0.07 0.03
0.12 0.03 0.08 0.20 0.06 0.00 0.00 0.11 0.08 0.08
0.04 0.07 0.11 0.16 0.04 0.00 0.05 0.13 0.00 0.04
0.00 0.00 0.04 0.16 0.00 0.08 0.00 0.00 0.06 0.00
0.03 0.00 0.00 0.04 0.00 0.04 0.00 0.04 0.04 0.04
0.11 0.00 0.16 0.12 0.04 0.04 0.00 0.13 0.08 0.04
0.03 0.00 0.09 0.09 0.00 0.03 0.00 0.03 0.03 0.00
0.00 0.42 0.04 0.04 0.00 0.00 0.16 0.00 0.08 0.07
0.00 0.00 0.04 0.00 0.07 0.00 0.04 0.00 0.08 0.00
0.20 0.00 0.15 0.00 0.04 0.08 0.00 0.00 0.26 0.03
0.48 0.00 0.00 0.00 0.04 0.05 0.00 0.00 0.23 0.00
0.04 0.04 0.46 0.07 0.00 0.04 0.04 0.13 0.00 0.00
0
1
2
4
6
7
8
9
10
11
12
13
14
30
0
1
3
4
5
6
7
8
9
10
OBLONG

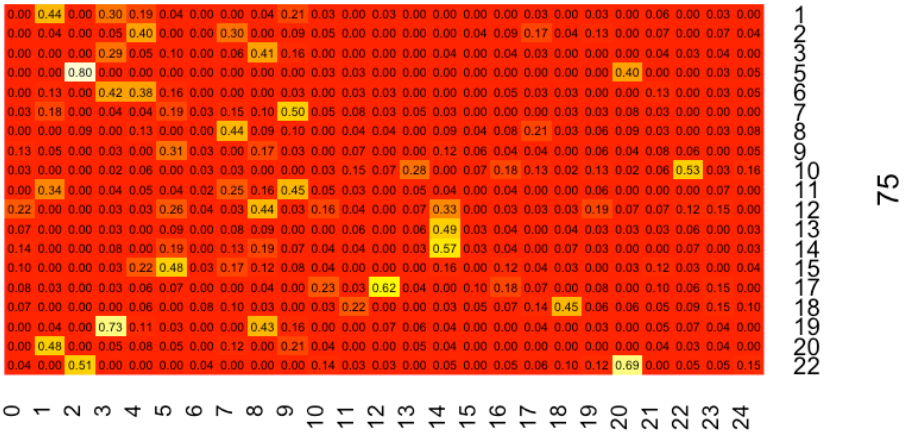

Color Key
and Histogram
Count
0
0
0.2
0.4
0.6
Value
0.00 0.73 0.00 0.00 0.05 0.00 0.00 0.05 0.00 0.54 0.00 0.00 0.00 0.00 0.00 0.00 0.03 1
0.00 0.00 0.44 0.10 0.04 0.11 0.03 0.00 0.14 0.07 0.13 0.05 0.22 0.03 0.00 0.00 0.00 2
0.00 0.06 0.33 0.04 0.04 0.04 0.00 0.00 0.09 0.04 0.11 0.06 0.32 0.00 0.00 0.00 0.03 5
0.00 0.00 0.35 0.04 0.00 0.28 0.03 0.00 0.04 0.03 0.15 0.04 0.16 0.04 0.09 0.00 0.06 6
0.00 0.03 0.00 0.04 0.00 0.00 0.00 0.00 0.07 0.05 0.04 0.04 0.00 0.03 0.00 0.00 0.07 8
0.04 0.00 0.00 0.00 0.11 0.04 0.00 0.04 0.04 0.04 0.11 0.36 0.20 0.03 0.20 0.03 0.24 11
0.00 0.03 0.30 0.00 0.04 0.10 0.00 0.00 0.07 0.08 0.17 0.12 0.58 0.13 0.00 0.00 0.08 12
0.03 0.06 0.02 0.00 0.00 0.00 0.04 0.03 0.15 0.03 0.33 0.11 0.07 0.00 0.03 0.04 0.00 13
0.00 0.46 0.00 0.00 0.09 0.06 0.00 0.05 0.04 0.73 0.00 0.06 0.08 0.00 0.00 0.00 0.03 14
0.04 0.00 0.10 0.03 0.03 0.33 0.00 0.00 0.00 0.00 0.12 0.15 0.00 0.00 0.30 0.00 0.15 15
0.00 0.00 0.00 0.00 0.00 0.10 0.03 0.04 0.00 0.00 0.00 0.03 0.12 0.00 0.28 0.08 0.09 17
0.00 0.00 0.00 0.12 0.04 0.04 0.00 0.04 0.17 0.08 0.03 0.12 0.00 0.04 0.04 0.17 0.08 18
0.07 0.00 0.07 0.24 0.04 0.00 0.09 0.00 0.23 0.00 0.03 0.04 0.03 0.58 0.00 0.04 0.03 19
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
50
DOME

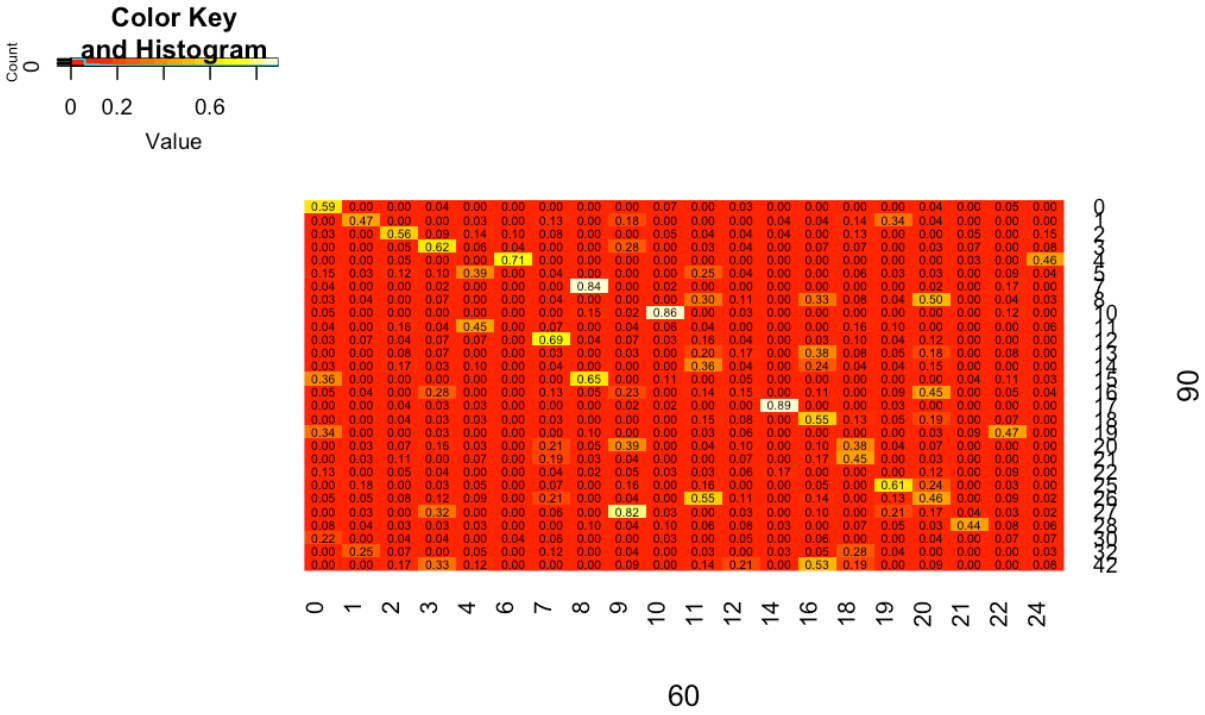

Color Key
and Histogram
Count
0
0
0.2
0.4
0.6
Value
0.20 0.00 0.00 0.17 0.00 0.04 0.09 0.03 0.03 0.00 0.07 0.00 0.31 0
0.04 0.04 0.18 0.09 0.22 0.00 0.07 0.00 0.00 0.00 0.00 0.00 0.00 1
0.00 0.66 0.00 0.00 0.00 0.00 0.00 0.00 0.45 0.00 0.00 0.00 0.00 2
0.06 0.00 0.68 0.14 0.14 0.10 0.33 0.02 0.00 0.00 0.00 0.00 0.17 3
0.03 0.00 0.29 0.57 0.18 0.10 0.00 0.00 0.05 0.04 0.00 0.00 0.03 4
0.51 0.00 0.03 0.20 0.00 0.04 0.07 0.06 0.08 0.00 0.00 0.00 0.21 5
0.03 0.00 0.00 0.02 0.00 0.07 0.00 0.72 0.00 0.05 0.00 0.02 0.00 6
0.04 0.04 0.04 0.28 0.37 0.00 0.08 0.00 0.08 0.04 0.04 0.00 0.04 7
0.32 0.00 0.25 0.04 0.13 0.00 0.56 0.00 0.03 0.04 0.06 0.00 0.38 8
0.05 0.03 0.20 0.14 0.39 0.00 0.19 0.00 0.03 0.00 0.00 0.00 0.04 9
0.04 0.08 0.00 0.11 0.04 0.08 0.00 0.00 0.12 0.00 0.24 0.17 0.09 10
0.00 0.03 0.00 0.03 0.00 0.07 0.00 0.06 0.00 0.15 0.04 0.00 0.00 11
0.07 0.00 0.07 0.00 0.00 0.07 0.14 0.09 0.00 0.00 0.11 0.42 0.04 12
0.00 0.00 0.03 0.03 0.00 0.33 0.03 0.25 0.00 0.03 0.00 0.04 0.00 13
0.23 0.03 0.03 0.06 0.07 0.00 0.22 0.00 0.04 0.00 0.16 0.00 0.20 14
0.00 0.00 0.03 0.03 0.00 0.12 0.07 0.11 0.00 0.03 0.06 0.15 0.00 15
0.11 0.00 0.00 0.00 0.00 0.23 0.12 0.03 0.00 0.03 0.03 0.03 0.00 16
0.12 0.00 0.04 0.07 0.00 0.13 0.00 0.15 0.00 0.10 0.04 0.00 0.00 17
0.00 0.00 0.05 0.03 0.00 0.00 0.04 0.03 0.00 0.36 0.00 0.05 0.03 18
0.03 0.07 0.00 0.00 0.00 0.06 0.07 0.00 0.08 0.00 0.06 0.05 0.08 19
0.00 0.46 0.03 0.07 0.00 0.00 0.00 0.03 0.63 0.05 0.04 0.00 0.03 20
0.17 0.00 0.03 0.03 0.00 0.04 0.03 0.06 0.04 0.09 0.00 0.09 0.00 21
0.00 0.00 0.00 0.06 0.00 0.37 0.12 0.06 0.03 0.04 0.03 0.00 0.04 22
0.00 0.00 0.00 0.00 0.00 0.03 0.00 0.02 0.03 0.03 0.13 0.06 0.11 23
0.08 0.05 0.03 0.03 0.00 0.09 0.15 0.03 0.06 0.04 0.04 0.06 0.08 24
0 1 2 4 6 7 8 9 10 11 12 13 14
S
30

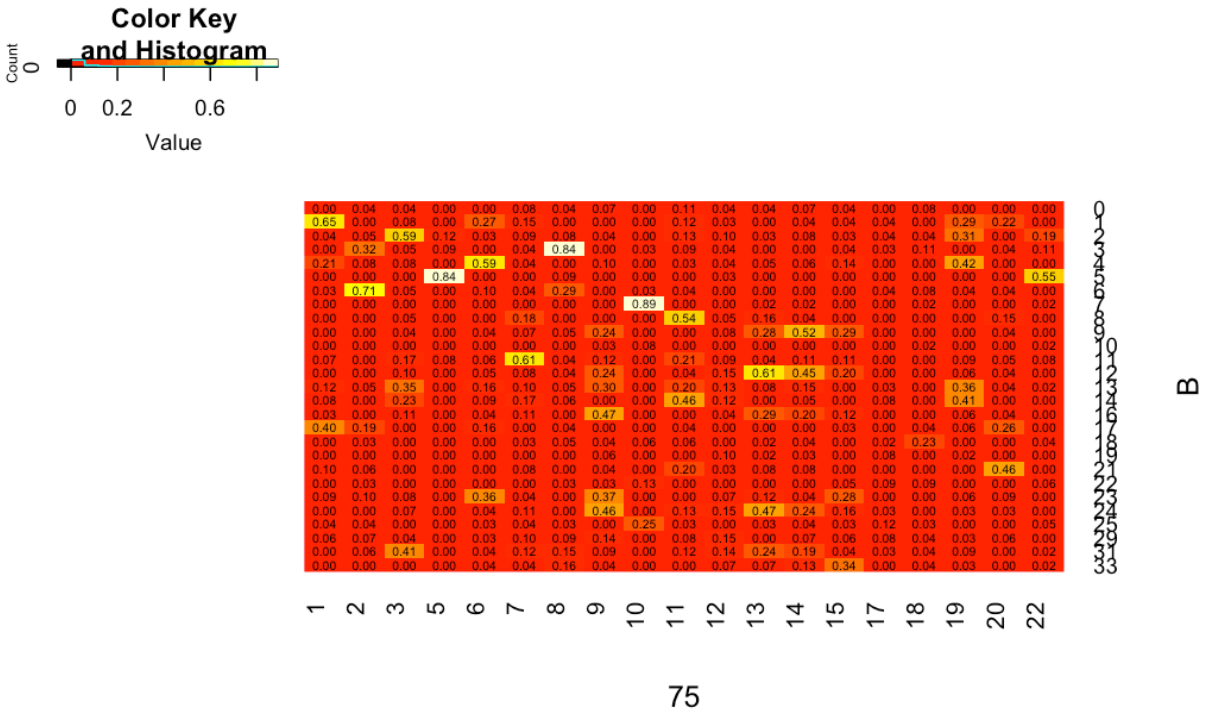

Color Key
and Histogram
Count
0
0
0.2
0.6
Value

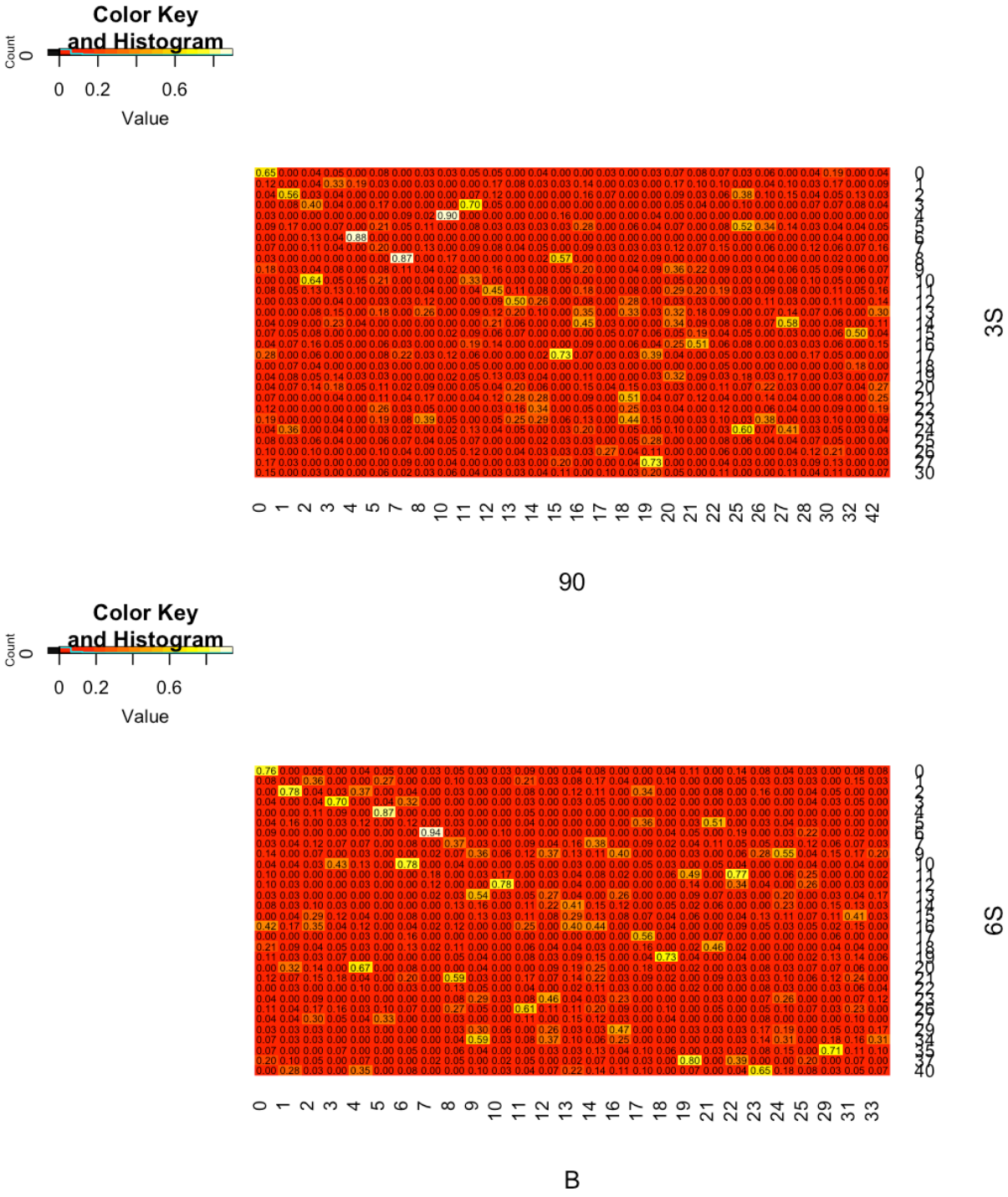

0.00 0.00 0.00 0.00 0.00 0.03 0.00 0.07 0.00 0.00 0.00 0.07 0.04
0.00 0.39 0.11 0.24 0.00 0.05 0.19 0.00 0.00 0.00 0.00 0.00 0.03
0.05 0.16 0.63 0.00 0.04 0.07 0.22 0.05 0.10 0.08 0.00 0.00 0.00
0.00 0.08 0.08 0.17 0.08 0.08 0.12 0.04 0.07 0.54 0.00 0.00 0.00
0.00 0.19 0.30 0.08 0.03 0.13 0.65 0.03 0.06 0.06 0.00 0.00 0.00
0.83 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.51 0.00 0.00 0.11 0.00
0.00 0.17 0.14 0.11 0.00 0.16 0.12 0.05 0.00 0.16 0.04 0.04 0.00
0.00 0.00 0.00 0.00 0.00 0.03 0.00 0.00 0.00 0.03 0.07 0.00 0.02
0.00 0.13 0.05 0.59 0.00 0.07 0.11 0.02 0.00 0.53 0.07 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.06 0.02 0.00 0.64 0.00 0.02
0.00 0.00 0.04 0.04 0.00 0.65 0.33 0.04 0.09 0.17 0.09 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.36 0.03 0.10 0.00 0.32 0.00 0.03 0.03
0.00 0.00 0.00 0.00 0.84 0.00 0.03 0.12 0.00 0.03 0.03 0.03 0.03
0.00 0.04 0.12 0.00 0.00 0.12 0.00 0.00 0.03 0.10 0.00 0.00 0.00
0.00 0.38 0.38 0.09 0.03 0.00 0.18 0.03 0.03 0.08 0.00 0.03 0.00
0.00 0.13 0.03 0.57 0.03 0.11 0.29 0.00 0.00 0.00 0.06 0.04 0.04
0.00 0.00 0.00 0.14 0.00 0.37 0.04 0.13 0.04 0.27 0.07 0.04 0.00
0.03 0.00 0.00 0.00 0.00 0.00 0.00 0.06 0.06 0.00 0.00 0.23 0.41
0.00 0.00 0.00 0.00 0.00 0.04 0.00 0.04 0.00 0.00 0.14 0.08 0.00
0.48 0.00 0.00 0.00 0.00 0.03 0.00 0.00 0.76 0.03 0.00 0.12 0.00
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 24
1 2 5 6 8 11 12 13 14 15 17 18 19
60
50


Color Key
and Histogram
Count
0
0
0.2
0.4
0.6
Value


0.00 0.44 0.00 0.30 0.19 0.04 0.00 0.00 0.04 0.21 0.03 0.00 0.03 0.00 0.00 0.00 0.00 0.03 0.00 0.03 0.00 0.06 0.00 0.03 0.00
0.00 0.04 0.00 0.05 0.40 0.00 0.00 0.30 0.00 0.09 0.05 0.00 0.00 0.00 0.00 0.04 0.09 0.17 0.04 0.13 0.00 0.07 0.00 0.07 0.04
0.00 0.00 0.00 0.29 0.05 0.10 0.00 0.06 0.41 0.16 0.00 0.00 0.00 0.00 0.04 0.00 0.04 0.03 0.00 0.00 0.00 0.04 0.03 0.04 0.00
0.00 0.00 0.80 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.03 0.03 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.40 0.00 0.00 0.03 0.05
0.00 0.13 0.00 0.42 0.38 0.16 0.00 0.00 0.00 0.03 0.00 0.03 0.03 0.00 0.00 0.00 0.05 0.03 0.03 0.00 0.00 0.13 0.00 0.03 0.05
0.03 0.18 0.00 0.04 0.04 0.19 0.03 0.15 0.10 0.50 0.05 0.08 0.03 0.05 0.03 0.00 0.00 0.00 0.03 0.03 0.08 0.03 0.00 0.00 0.00
0.00 0.00 0.09 0.00 0.13 0.00 0.00 0.44 0.09 0.10 0.00 0.04 0.04 0.00 0.09 0.04 0.08 0.21 0.03 0.06 0.09 0.03 0.00 0.03 0.08
0.13 0.05 0.00 0.03 0.00 0.31 0.03 0.00 0.17 0.03 0.00 0.07 0.00 0.00 0.12 0.06 0.04 0.04 0.00 0.06 0.04 0.08 0.06 0.00 0.05
0.03 0.00 0.00 0.02 0.06 0.00 0.03 0.03 0.00 0.00 0.03 0.15 0.07 0.28 0.00 0.07 0.18 0.13 0.02 0.13 0.02 0.06 0.53 0.03 0.16
0.00 0.34 0.00 0.04 0.05 0.04 0.02 0.25 0.16 0.45 0.05 0.03 0.00 0.05 0.04 0.00 0.04 0.00 0.00 0.06 0.00 0.00 0.00 0.07 0.00
0.22 0.00 0.00 0.03 0.03 0.26 0.04 0.03 0.44 0.03 0.16 0.04 0.00 0.07 0.33 0.00 0.03 0.03 0.03 0.19 0.07 0.07 0.12 0.15 0.00
0.07 0.00 0.00 0.03 0.00 0.09 0.00 0.08 0.09 0.00 0.00 0.06 0.00 0.06 0.49 0.03 0.04 0.00 0.04 0.03 0.03 0.03 0.06 0.00 0.03
0.14 0.00 0.00 0.08 0.00 0.19 0.00 0.13 0.19 0.07 0.04 0.04 0.00 0.03 0.57 0.03 0.04 0.00 0.07 0.03 0.00 0.00 0.07 0.00 0.03
0.10 0.00 0.00 0.03 0.22 0.48 0.03 0.17 0.12 0.08 0.04 0.00 0.00 0.00 0.16 0.00 0.12 0.04 0.03 0.00 0.03 0.12 0.03 0.00 0.04
0.08 0.03 0.00 0.03 0.06 0.07 0.00 0.00 0.04 0.00 0.23 0.03 0.62 0.04 0.00 0.10 0.18 0.07 0.00 0.08 0.00 0.10 0.06 0.15 0.00
0.07 0.00 0.00 0.00 0.06 0.00 0.08 0.10 0.03 0.00 0.03 0.22 0.00 0.00 0.03 0.05 0.07 0.14 0.45 0.06 0.06 0.05 0.09 0.15 0.10
0.00 0.04 0.00 0.73 0.11 0.03 0.00 0.00 0.43 0.16 0.00 0.00 0.07 0.06 0.04 0.00 0.00 0.04 0.00 0.03 0.00 0.05 0.07 0.04 0.00
0.00 0.48 0.00 0.05 0.08 0.05 0.00 0.12 0.00 0.21 0.04 0.00 0.00 0.05 0.04 0.00 0.00 0.00 0.00 0.00 0.00 0.04 0.03 0.04 0.00
0.04 0.00 0.51 0.00 0.00 0.00 0.04 0.00 0.00 0.00 0.14 0.03 0.03 0.00 0.05 0.00 0.05 0.06 0.10 0.12 0.69 0.00 0.05 0.05 0.15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
1 2 3 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 22
75
S


Color Key
and Histogram
Count
0
0
0.2
0.6
Value
90
60


Color Key
and Histogram
Count
0
0
0.2
0.6
Value
B
75



Connect modules using the overlap scores calculated above

Build tables that record potential connections

For each module, find its most overlaped module in each of the two previous stages. Only modules with >20% overlaps are taken into account.

```
## for each module at one stage, want to find max correlated one in the two
## previous stages
```

```
connect_module <- function(thres1 = 0.15, thres2 = 0.25, G_cor_use, G_cor_use2) {
    G_connect <- list()
    for (i in 1:(length(stages) - 1)) {
        stage = stages[i]
        stage_next = stages[i + 1]
        G_cor_stage = G_cor_use[[stage]]
        Max_pre = apply(G_cor_stage, 1, order)
        Max_pre_ind = Max_pre[dim(Max_pre)[1], ]
        Max_pre_M = colnames(G_cor_stage)[Max_pre_ind]
        Max_value = apply(G_cor_stage, 1, max)
        has_pre_ind = which(Max_value > thres1)
        has_pre_M = rownames(G_cor_stage)[has_pre_ind]
        if (i == 1) {
            G_connect[[stage_next]] = data.frame(matrix(NA, nrow = 1, ncol = dim(G_cor_stage)[1]),
                row.names = stage)
            colnames(G_connect[[stage_next]]) = rownames(G_cor_stage)
            G_connect[[stage_next]][, has_pre_M] = Max_pre_M[has_pre_ind]
            G_connect[[stage_next]] = G_connect[[stage_next]][, has_pre_M]
        } else {
            stage_pre = stages[i - 1]
            G_cor_stage2 = G_cor_use2[[stage_pre]]
            all_M = union(rownames(G_cor_stage2), rownames(G_cor_stage))
            G_connect[[stage_next]] = data.frame(matrix(NA, nrow = 2, ncol = length(all_M)),
                row.names = c(stage, stage_pre))
            colnames(G_connect[[stage_next]]) = all_M
            G_connect[[stage_next]][1, has_pre_M] = Max_pre_M[has_pre_ind]
            G_cor_stage = G_cor_use2[[stage_pre]]
            Max_pre = apply(G_cor_stage, 1, order)
            Max_pre_ind = Max_pre[dim(Max_pre)[1], ]
            Max_pre_M = colnames(G_cor_stage)[Max_pre_ind]
            Max_value = apply(G_cor_stage, 1, max)
            has_pre_ind = which(Max_value > thres2)
            has_pre_M2 = rownames(G_cor_stage)[has_pre_ind]
            G_connect[[stage_next]][2, has_pre_M2] = Max_pre_M[has_pre_ind]
            G_connect[[stage_next]] = G_connect[[stage_next]][, union(has_pre_M,
                has_pre_M2)]
        }
    }
    return(G_connect)
}
G_int_connect = connect_module(G_cor_use = G_int_use, G_cor_use2 = G_int2, thres1 = 0.2,
    thres2 = 0.2)
```

Build an adjacency matrix to record the final connections between modules

We start from modules in the oldest stage (6-somites). Each module is first connected to its most overlaped module in the immediate previous stage. If no potential connection is recorded (in `G_int_connect`) for the immediate previous stage, it will then be connected to the module recorded for the stage earlier (if there is one). When the overlap between a module and its most overlapped module in the immediate previous stage is less than 30%, and at the same time it has more than 50% overlap with its most overlapped module two stages earlier, we then directly connect this module to the more previous module, and cut its connection to the one in the immidiate previous stage.

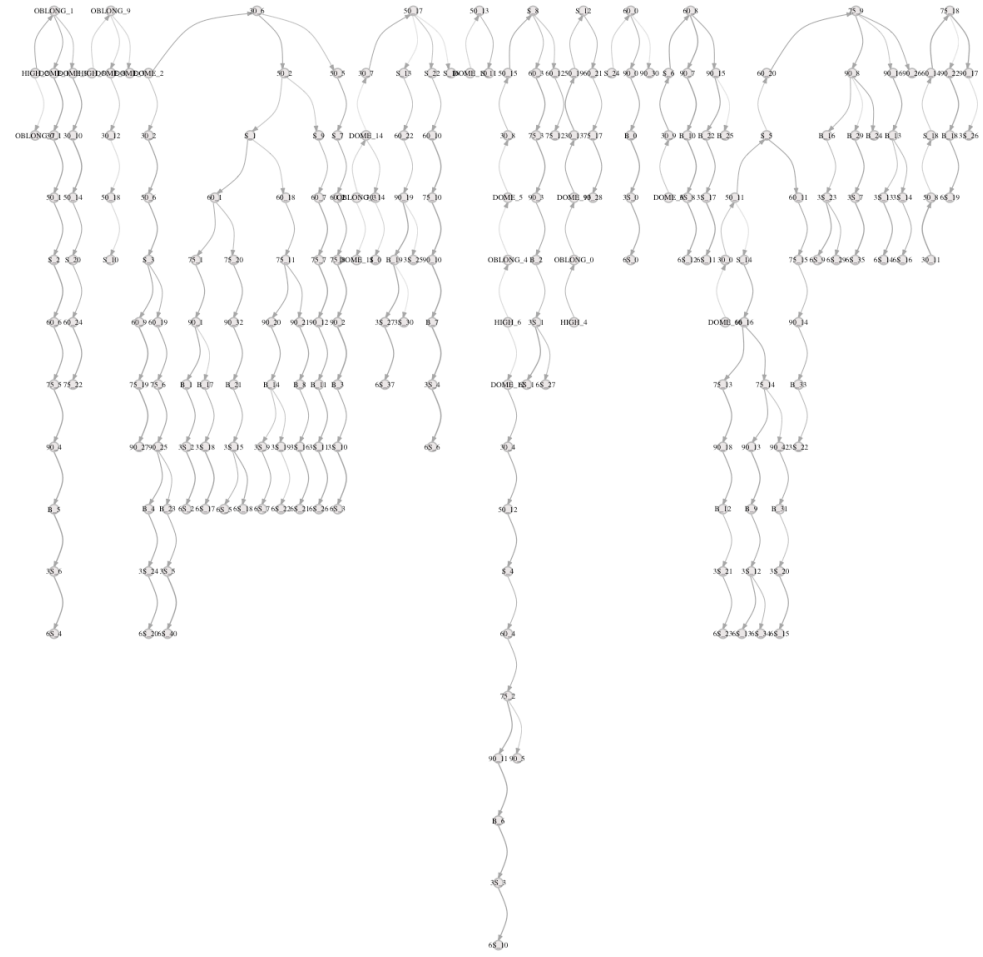
```
build_netM <- function(G_connect, G_cor_use, G_cor_use2, thres = NULL, thres_pre = NULL) {
    nodes_names = c()
    for (i in 1:(length(stages) - 1)) {
        stage = stages[i + 1]
        G_ans = G_connect[[stage]]
        nodes_names = union(nodes_names, paste0(stage, "_", colnames(G_ans)))
        nodes_names = union(nodes_names, paste0(stages[i], "_", G_ans[stages[i],
            which(!is.na(G_ans[stages[i], ]))]))
        if (i > 1) {
            nodes_names = union(nodes_names, paste0(stages[i - 1], "_", G_ans[stages[i -
                1], which(!is.na(G_ans[stages[i - 1], ]))]))
        }
    }
    num_nodes = length(nodes_names)
    net_M = matrix(0, ncol = num_nodes, nrow = num_nodes)
    rownames(net_M) = nodes_names
```

```
    colnames(net_M) = nodes_names

    for (i in 1:(length(stages) - 1)) {
        stage_pre = stages[i]
        stage = stages[i + 1]
        G_ans = G_connect[[stage]]
        for (j in colnames(G_ans)) {
            to_name = paste0(stage, "_", j)
            if (!is.na(G_ans[stage_pre, j])) {
                from_M = G_ans[stage_pre, j]
                from_name = paste0(stage_pre, "_", from_M)
                ## get the correlation score to put in the connection matirx
                net_M[from_name, to_name] = G_cor_use[[stage_pre]][j, from_M]
            }
            if (i != 1) {
                stage_pre2 = stages[i - 1]
                if (!is.na(G_ans[stage_pre2, j])) {
                  from_M2 = G_ans[stage_pre2, j]
                  from_name2 = paste0(stage_pre2, "_", from_M2)
                  if (is.na(G_ans[stage_pre, j])) {
                    net_M[from_name2, to_name] = G_cor_use2[[stage_pre2]][j, from_M2]
                  } else if (!is.null(thres)) {
                    G_cor = G_cor_use[[stage_pre]][j, from_M]
                    G_cor_pre = G_cor_use2[[stage_pre2]][j, from_M2]
                    if (G_cor < thres && G_cor_pre > thres_pre) {
                      print(paste0("add ", from_name2, " to ", to_name))
                      net_M[from_name2, to_name] = G_cor_use2[[stage_pre2]][j, from_M2]
                      print(paste0("delete ", from_name, " to ", to_name))
                      net_M[from_name, to_name] = 0
                    }
                  }
                }
            }
        }
    }
    return(net_M)
}

net_int = build_netM(G_int_connect, G_int_use, G_int2, thres = 0.3, thres_pre = 0.5)
```

```
## [1] "add B_18 to 6S_19"
## [1] "delete 3S_11 to 6S_19"
```

Visualize the connections using igraph

```
draw.net = function(net_M, circular = T, label.size = 0.5) {
    ind_use = union(which(apply(net_M, 1, sum) > 0), which(apply(net_M, 2, sum) >
        0))
    net_M = net_M[ind_use, ind_use]
    net = graph.adjacency(net_M, mode = "directed", weighted = TRUE, diag = TRUE)
    plot(net, vertex.label = V(net)$name, vertex.label.color = "black", edge.width = E(net)$weight *
        1, edge.arrow.size = 0.2, edge.curved = TRUE, vertex.size = 2, vertex.label.cex = label.size,
        vertex.color = "snow2", vertex.frame.color = "gray", layout = layout_as_tree(net,
            mode = "all", circular = circular))
}
draw.net(net_int, circular = F, label.size = 0.37)
```



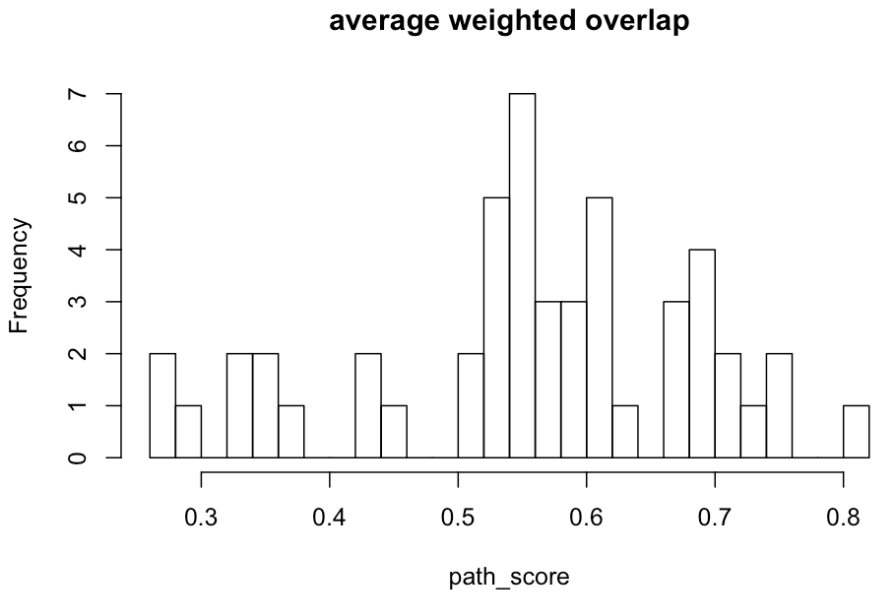
Trim path with poor quality

```
get_downstream <- function(net_M, start_M, exclude = c("")) {
    all_ds = c(start_M)
    M_ds = colnames(net_M)[which(net_M[start_M, ] > 0)]
    M_ds = M_ds[which(!M_ds %in% exclude)]
    if (length(M_ds) > 0) {
        all_ds = unique(c(all_ds, M_ds))
        for (M_d in M_ds) {
            all_ds = unique(c(all_ds, get_downstream(net_M, M_d, exclude = exclude)))
        }
    }
    return(all_ds)
}

get_upstream <- function(net_M, start_M, exclude = c(""), mean_score = F, start_score = 0,
```

```
    start_num_ans = 0) {
    all_as = c(start_M)
    M_as = rownames(net_M)[which(net_M[, start_M] > 0)]
    M_as = M_as[which(!M_as %in% exclude)]
    num_ans = start_num_ans
    tot_score = start_score
    if (length(M_as) > 0) {
        all_as = unique(c(all_as, M_as))
        num_ans = num_ans + length(M_as)
        # print(num_ans)
        tot_score = tot_score + sum(net_M[M_as, start_M])
        # print(tot_score)

        for (M_a in M_as) {
            if (mean_score) {
                in_result_list = get_upstream(net_M, M_a, exclude = exclude, mean_score = T,
                  start_score = tot_score, start_num_ans = num_ans)
                all_as = unique(c(all_as, in_result_list$upstream))
                # print(all_as) print(in_result_list$score)
                tot_score = in_result_list$score[1]
                num_ans = in_result_list$score[2]
            } else {
                all_as = unique(c(all_as, get_upstream(net_M, M_a, exclude = exclude)))
            }
        }
    }
    if (mean_score) {
        return_list = list()
        return_list$upstream = all_as
        return_list$score = c(tot_score, num_ans)
        return(return_list)
    } else {
        return(all_as)
    }
}

calc_path_qual <- function(net_M, path = "all", exclude = c("")) {
    ## calculate the mean overlap level along the path end at the specified node(s)
    if (path == "all") {
        end_nodes = rownames(net_M)[which(apply(net_M, 1, max) == 0)]
    } else {
        end_nodes = path
    }
    score_vec = c(1:length(end_nodes)) * 0
    names(score_vec) = end_nodes
    for (node in end_nodes) {
        node_score = get_upstream(net_M, node, mean_score = T, exclude = exclude)
        score_vec[node] = node_score$score[1]/node_score$score[2]
    }
    return(score_vec)
}
```

Calculate the average overlap score along each chain of connected gene modules

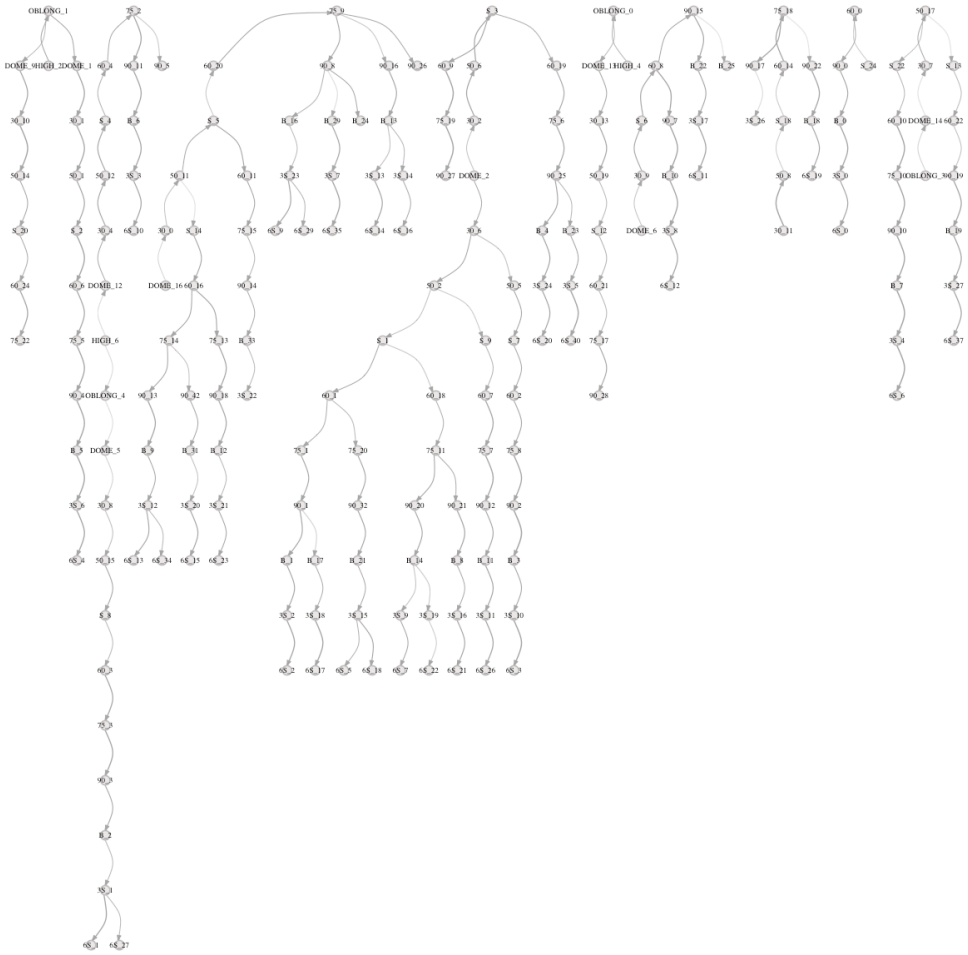
```
path_score = calc_path_qual(net_int)
hist(path_score, breaks = 30, main = "average weighted overlap")
```



Keep only the paths with >0.45 average weighted overlap.

Most of the path with <0.45 average overlap were short or consist of either ubiquitous or lowly expressed genes.

```
end_nodes_good = names(path_score[path_score >= 0.45])
all_nodes_good = c()
for (node in end_nodes_good) {
    all_nodes_good = c(all_nodes_good, get_upstream(net_int, node))
}
all_nodes_good = unique(all_nodes_good)

net_int_good = net_int[all_nodes_good, all_nodes_good]
draw.net(net_int_good, circular = F, label.size = 0.37)
```



Save this adjacency matrix for more customed visualization in *yed*

```
write.csv(net_int_good, file = "../Module Tree/knit_final_adj_M.csv")
```

Save connected module information for overlaying on URD tree

For each module at the end (oldest developmental stage) of a connected chain, find all its upstream modules, and store them as an entry in one list

```
all_end_nodes = rownames(net_int_good)[which(apply(net_int_good, 1, sum) == 0)]
all_lineages <- list()
for (end_node in all_end_nodes) {
    all_lineages[[end_node]] = get_upstream(net_int_good, end_node)
```

```
}
save(all_lineages, file = "../Module Tree/knit_module_lineages.Robj")
```

For modules that are in the same connected chain, sum up their levels in each cell to represent the expression of that lineage program. This results is a lineage by cell matrix

```
all_cells = c()
all_genes = c()
for (stage in stages) {
    C_use = DS_C_use[[stage]]
    all_cells = c(all_cells, colnames(C_use))
    G_use = DS_G_use[[stage]]
    all_genes = c(all_genes, rownames(G_use))
}

all_genes = unique(all_genes)
all_Ms = rownames(net_int_good)
allM_allCell = data.frame(matrix(0, ncol = length(all_cells), nrow = length(all_Ms)),
    row.names = all_Ms)
allGene_allM = data.frame(matrix(0, ncol = length(all_Ms), nrow = length(all_genes)),
    row.names = all_genes)
colnames(allM_allCell) = all_cells
colnames(allGene_allM) = all_Ms
## look stage by stage, fill in the expression matrix with MAX NORMALIZED gene
## module expression
for (stage in stages) {
    G_use = DS_G_use[[stage]]
    G.max = apply(G_use, 2, max)
    G_norm = sweep(G_use, 2, G.max, "/")  ## now each module's top gene has weight 1
    colnames(G_norm) = paste0(stage, "_", colnames(G_norm))
    M_use = intersect(colnames(G_norm), all_Ms)

    C_use = DS_C_use[[stage]]
    C.max = apply(C_use, 1, max)
    C_norm = sweep(C_use, 1, C.max, "/")
    rownames(C_norm) = paste0(stage, "_", rownames(C_norm))

    if (length(M_use) > 0) {
        ## fill in gene matrix
        allGene_allM[rownames(G_norm), M_use] = G_norm[rownames(G_norm), M_use]
        ## fill in cell matrix
        allM_allCell[M_use, colnames(C_use)] = C_norm[M_use, colnames(C_use)]
    }
}

lineage_cell = data.frame(matrix(0, ncol = length(all_cells), nrow = length(all_end_nodes)),
    row.names = all_end_nodes)
colnames(lineage_cell) = all_cells

# matrix to use: allM_allCell
for (lin in all_end_nodes) {
    lin_M = all_lineages[[lin]]
    if (length(setdiff(lin_M, all_Ms)) == 0) {
        ## sum up and add
        lineage_cell[lin, ] = apply(allM_allCell[lin_M, colnames(lineage_cell)],
            2, sum)
    } else {
        print(paste(lin, "has module(s) that are not in the table"))
    }
}
```

Re-name some of the rownames in the lineage by cell matrix based on their expression in URD lineage

```
lineage_names = rownames(lineage_cell)
lineage_names[which(lineage_names == "6S_0")] = "Housekeeping"
lineage_names[which(lineage_names == "6S_1")] = "Epidermis"
lineage_names[which(lineage_names == "6S_2")] = "PSM"
```

```
lineage_names[which(lineage_names == "6S_3")] = "PCP"
lineage_names[which(lineage_names == "6S_4")] = "EVL"
lineage_names[which(lineage_names == "6S_5")] = "SomiteForming"
lineage_names[which(lineage_names == "6S_6")] = "CellCycle"
lineage_names[which(lineage_names == "6S_7")] = "HeartPrimordium"
lineage_names[which(lineage_names == "6S_9")] = "HindbrainR3456"
lineage_names[which(lineage_names == "6S_10")] = "Notochord"
lineage_names[which(lineage_names == "6S_13")] = "OpticCup"
lineage_names[which(lineage_names == "6S_14")] = "NeuralCrest"
lineage_names[which(lineage_names == "6S_15")] = "Placode"
lineage_names[which(lineage_names == "6S_17")] = "Adaxial"
lineage_names[which(lineage_names == "6S_18")] = "Somite"
lineage_names[which(lineage_names == "6S_19")] = "NegativeRegulationRnaSynthesis"
lineage_names[which(lineage_names == "6S_20")] = "Tailbud2"
lineage_names[which(lineage_names == "6S_21")] = "CephalicMeso"
lineage_names[which(lineage_names == "6S_22")] = "Hematopoeitic_Pronephros"
lineage_names[which(lineage_names == "6S_23")] = "Midbrain"
lineage_names[which(lineage_names == "6S_26")] = "Endoderm"
lineage_names[which(lineage_names == "6S_27")] = "NonNeuralEctoderm"
lineage_names[which(lineage_names == "6S_29")] = "HindbrainR7_SpinalCord"
lineage_names[which(lineage_names == "6S_34")] = "Telencephalon"
lineage_names[which(lineage_names == "6S_35")] = "SpinalCord"
lineage_names[which(lineage_names == "6S_37")] = "CellCycle2"
lineage_names[which(lineage_names == "6S_40")] = "Tailbud"
lineage_names[which(lineage_names == "3S_22")] = "Diencephalon"
lineage_names[which(lineage_names == "3S_26")] = "ApoptoticLike"
lineage_names[which(lineage_names == "90_28")] = "PGC"
lineage_names[which(lineage_names == "75_22")] = "EVL2"

rownames(lineage_cell) = lineage_names
```

Save this lineage by cell table. Also save a table with all modules and their levels in each cell

```
write.csv(allM_allCell, file = "../Module Tree/knit_AllModuleByAllCell.csv")
write.csv(lineage_cell, file = "../Module Tree/knit_LineageByCell_ModuleSum.csv")
```

for each 50% module, get its down stream connected modules and save their expression in all cells in a matrix

```
all_Ms = rownames(net_int_good)
M_stages = unlist(lapply(all_Ms, function(x) unlist(strsplit(x, "_"))[1]))
M_ZF50_ind = which(M_stages == "50")
M_ZF50 = all_Ms[M_ZF50_ind]

ZF50_M_after <- list()
for (M in M_ZF50) {
    ZF50_M_after[[M]] = get_downstream(net_int_good, M)
}

not_in_oep_M = c("50_5", "50_2", "50_12")
all_in_oep = c()
all_not_oep = c()
ubi_M = c("50_8", "50_17")
for (ZF50_M in names(ZF50_M_after)) {
    if (ZF50_M %in% not_in_oep_M) {
        all_not_oep = c(all_not_oep, ZF50_M_after[[ZF50_M]])
    } else {
        if (!ZF50_M %in% ubi_M) {
            all_in_oep = c(all_in_oep, ZF50_M_after[[ZF50_M]])
        }
    }
}

all_50M_NormSum = data.frame(matrix(0, ncol = dim(allM_allCell)[2], nrow = length(ZF50_M_after)),
    row.names = names(ZF50_M_after))
colnames(all_50M_NormSum) = colnames(allM_allCell)

for (name in names(ZF50_M_after)) {
```

```
    all_50M_NormSum[name, ] = apply(allM_allCell[ZF50_M_after[[name]], ], 2, sum)
}

oep_M = data.frame(matrix(0, ncol = dim(allM_allCell)[2], nrow = 2), row.names = c("In_oep",
    "Not_in_oep"))
colnames(oep_M) = colnames(allM_allCell)
oep_M["In_oep", ] = apply(all_50M_NormSum[intersect(all_in_oep, rownames(all_50M_NormSum)),
    ], 2, sum)
oep_M["Not_in_oep", ] = apply(all_50M_NormSum[not_in_oep_M, ], 2, sum)
```

save tables

```
write.csv(all_50M_NormSum, file = "../Module Tree/knit_ZF50_allModule_maxNormSum.csv")
write.csv(oep_M, file = "../Module Tree/knit_ZF50_OEPM_maxNormSum.csv")
```

save the top 25 genes for each module

```
top_25genes <- list()
for (M in colnames(allGene_allM)) {
    top_25genes[[M]] = rownames(allGene_allM)[order(allGene_allM[, M], decreasing = T)[1:25]]
}
save(top_25genes, file = "../Module Tree/knit_Module_top_25genes.Robj")
```

find and save member genes for each module using a mixture model

```
library(mixtools)
## first build a mixture model with gaussian mixture then select the genes with
## higher posterior for the distribution with higher mu return the list of genes
top_genes <- list()
thres = 0.15
# par(mfrow=c(3,3))
for (M in colnames(allGene_allM)) {
    genes_use = rownames(allGene_allM)[which(allGene_allM[, M] > thres)]
    vec = as.numeric(allGene_allM[allGene_allM[, M] > thres, M])
    mixmdl = normalmixEM(as.numeric(allGene_allM[allGene_allM[, M] > thres, M]),
        mean.constr = c(mean(vec[which(vec < 0.4)]), mean(vec[which(vec > 0.5)])),
        lambda = c(19, 1), epsilon = 1e-05)
    low_dist = order(mixmdl$mu)[1]
    high_dist = order(mixmdl$mu)[2]
    high_gen_ind = which(mixmdl$posterior[, high_dist] - mixmdl$posterior[, low_dist] >=
        0)
    low_weigh = min(as.numeric(allGene_allM[genes_use[high_gen_ind], M]))
    top_genes[[M]] = rownames(allGene_allM)[which(allGene_allM[, M] > low_weigh)]
    # plot(mixmdl,which=2) title(main=paste0('\n\n',M,',
    # ',as.character(length(top_genes[[M]]))))
}

## save the list
save(top_genes, file = "./Result_obj/knit_Module_top_genes_MixEM.Robj")
```

Batch module removal and Clustering analysis of SMART-seq dataset

```
library("knitr")
opts_chunk$set(tidy.opts=list(width.cutoff=80),tidy=TRUE,dev="png",dpi=150)
```

Read in the NMF result object

NMF was run using function `NMF` from `sklearn.decomposition` in Python *scikit-learn* library. The results were then integrated into an R object, which we read in below. We varied the number of modules (`n_components` argument in NMF function) from 5 to 25, and eventually chose to use the result from K=18, because it resulted in a low inconsistency and a high cophenetic coefficient when repeated 10 times with random initial conditions.

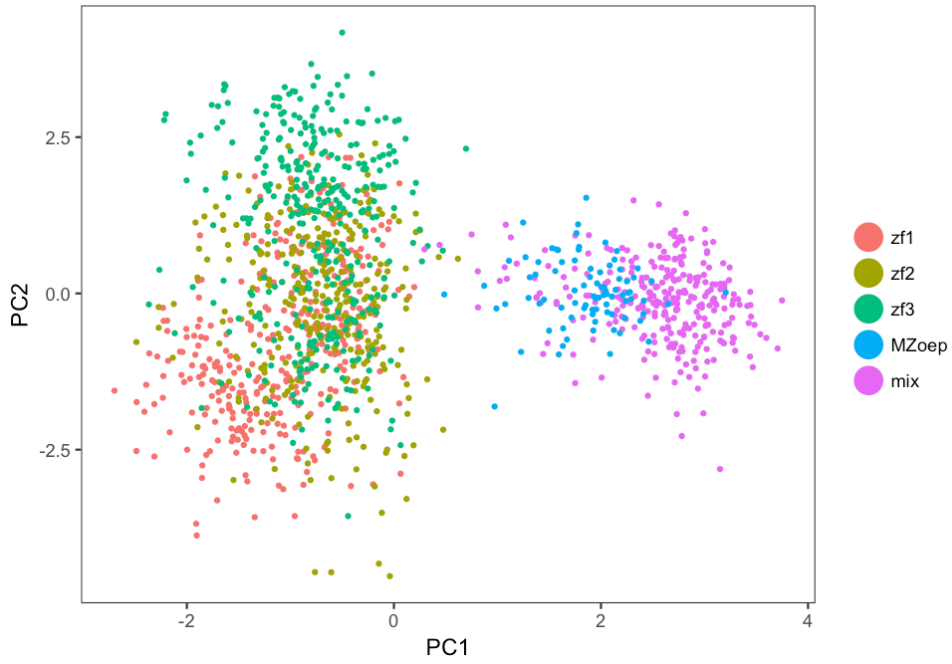
```
load_obj <- function(file.path) {
    temp.space <- new.env()
    obj <- load(file.path, temp.space)
    obj2 <- get(obj, temp.space)
    rm(temp.space)
    return(obj2)
}
nmf_res = load_obj("NMF/Results/P2_use/result_tbls.Robj")
nmf_K18 = nmf_res$`K=18`$rep0
```

`nmf_K18` contains the genes by modules matrix (G) and the modules by cells matrix (C) resulted from running NMF with `n_components=18`.

Find gene modules that are good predictors for experimental batches

First look at the PCA plot for all transcriptomes with all gene modules (using matrix C)

```
library("Seurat")
ALL_C18 = new("seurat", raw.data = nmf_K18$C)
ALL_C18 = Setup(ALL_C18, project = "allC18", min.cells = 2, names.field = 1, names.delim = "_",
    do.logNormalize = F, is.expr = 0.01, min.genes = 1)
ident = ALL_C18@ident
levels(ident) <- c(levels(ident), "MZoep", "mix")
ident[grep("wt", ident)] = "mix"
ident[grep("oep", ident)] = "mix"
ident[grep("oep_p0", names(ident))] = "MZoep"
ALL_C18@ident = ident

ALL_C18 = PCA(ALL_C18, do.print = F, pcs.print = 3, genes.print = 6, pc.genes = rownames(ALL_C18@data))
PCAPlot(ALL_C18, 1, 2, pt.size = 0.75)
```



Then separate transcriptomes from the two genotypes (wild-type and MZoep) and find batch modules for each genotype

```
wt_cells = c(grep("wt", ALL_C18@cell.names), grep("zf", ALL_C18@cell.names))
oep_cells = c(grep("oep", ALL_C18@cell.names))
ALL_C18wt = SubsetData(ALL_C18, cells.use = ALL_C18@cell.names[wt_cells])
ALL_C18oep = SubsetData(ALL_C18, cells.use = ALL_C18@cell.names[oep_cells])

batch_modulewt = BatchGene(ALL_C18wt, idents.use = c("zf1", "zf2", "zf3"), genes.use = rownames(ALL_C18wt@data),
    auc.cutoff = 0.67)
batch_moduleoep = BatchGene(ALL_C18oep, idents.use = c("MZoep", "mix"), genes.use = rownames(ALL_C18oep@data),
    auc.cutoff = 0.67)

batch_modules18 = union(batch_moduleoep, batch_modulewt)
print("Batch modules:")
```

```
## [1] "Batch modules:"
print(batch_modules18)
```

```
## [1] "5"  "7"  "2"  "0"  "1"  "6"  "10" "13"
batch_genes = nmf_K18$top30genes[, paste0("Module.", batch_modules18)]
# knitr::kable(batch_genes,caption = 'Top 30 genes in batch modules')
print(xtable::xtable(batch_genes, caption = "Top 30 genes in batch modules"), type = "latex",
    scalebox = 0.55)
```

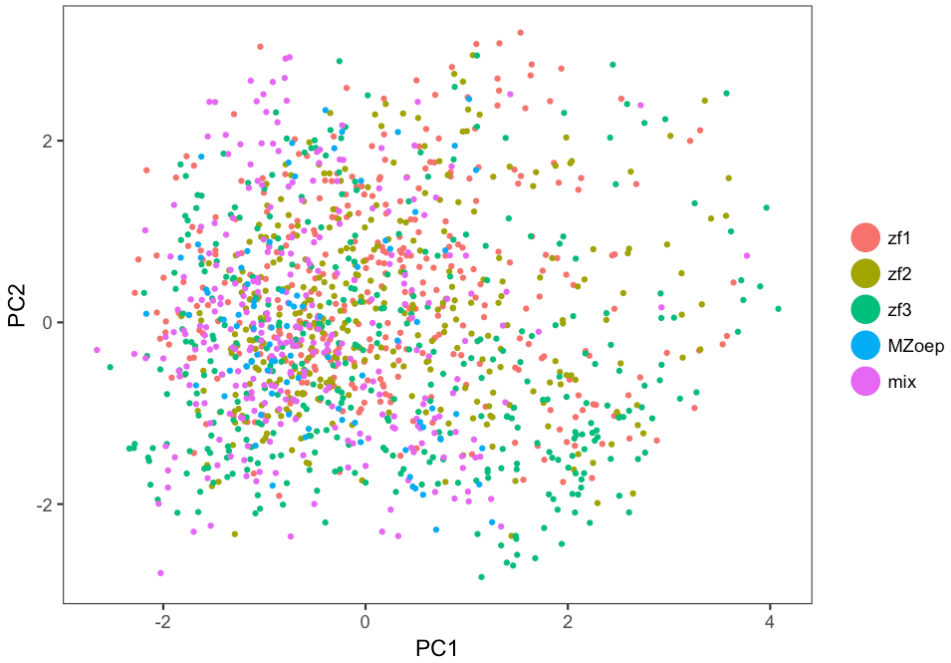
% latex table generated in R 3.4.0 by xtable 1.8-2 package % Tue Feb 20 12:00:15 2018

Generate PCA plot for all cells with all the non-batch gene modules

```
ALL_C18 = PCA(ALL_C18, do.print = F, pcs.print = 3, genes.print = 6, pc.genes = setdiff(rownames(ALL_C18@data),
    batch_modules18))
PCAPlot(ALL_C18, 1, 2, pt.size = 0.75)
```

	Module.5	Module.7	Module.2	Module.0	Module.1	Module.6	Module.10	Module.13
0	SI:DKEY-23A13.18	SI:CH73-195I19.3	HS3ST3B1B	ANO9A	ZGC:158463	BX901974.1	RBM4.3	ZGC:163040
1	SI:DKEY-23A13.14	TOP1	SI:DKEY-28A3.2	ZGC:173587	OTUD4	BX470115.2	SI:DKEYP-3B12.7	ZGC:194989
2	METAZOA_SRP	CABZ01084942.1	PHYHIP	UCP3	SI:CH211-195B11.4	SI:CH211-234C11.3	SI:DKEYP-3B12.6	SI:CH73-36P18.2
3	SI:DKEY-23A13.16	RAPGEF6	UPK1A	SI:CH211-113A14.24	SI:CH211-113A14.16	BX927193.1	TAF15	FP236812.1
4	ZGC:153405	ZGC:171775	ARID3A	ZNF1066	SI:DKEY-6D5.4	SI:DKEY-238O14.7	ALYREF	SI:DKEY-108K21.14
5	RN7SK	PROCA	KCTD5	HNRNPA0B	SI:CH1073-159D7.7	BX511215.1	METAZOA_SRP	SI:CH211-113A14.24
6	ZGC:173587	FNIP1	TRIM45	ZNF1070	SI:DKEY-23A13.4	ZNF1099	CHD8	SI:DKEY-108K21.21
7	SI:DKEY-261M9.12	NME2A	NCOA7	SI:DKEYP-3B12.6	SI:CH211-267E7.11	ZNF1031	ENSDART00000133990	SI:DKEY-261M9.11
8	SI:DKEY-261M9.19	SI:CH211-168H21.2	SI:DKEY-58J15.10	ZGC:113984	DUT	ZNF1040	SI:DKEY-56M15.3	CHD8
9	TRPC4A	GPRC5BB	RNF220B	SI:DKEYP-3B12.7	SI:DKEY-23A13.18	ZNF1125	HNRNPA1A	SI:DKEY-108K21.17
10	MPL	CR318603.2	SI:CH211-196H16.12	SI:CH73-368J24.14	P4HB	BX005448.2	ENSDART00000129497	B3GLCTB
11	SI:DKEY-108K21.26	AGRP2	SYNGAP1A	ZGC:110434	RPL6	SI:DKEY-14O1.14	BX324155.1	SPTBN1
12	MYO3A	DIABLOA	CNR1	SYNCRIP	NCALDA	SI:CH211-223A21.3	POLR2A	SLC7A1
13	CFD	PON1	GRB7	ENSDART00000129497	SI:DKEY-23A13.10	ZNF1074	HNRNPM	SI:CH211-113A14.15
14	RPP25	SI:CH211-241E1.5	JAG1B	RPL14	SI:CH211-272F15.5	ZNF1084	H3F3B.1	FP236812.2
15	SI:CH211-113A14.18	DDC	NRXN1B	SI:CH211-128N12.4	PPIB	ZNF1063	BX005448.2	ZGC:173652
16	SMC1A	MYO3A	TNFAIP8L2A	ZNF1068	GDF3	BX324179.3	TNPO2	ZGC:113984
17	SETD2	AGRP	CNNM1	SNRPC	BZW1A	BX324155.1	PRAM1	JARID2B
18	SI:DKEY-108K21.17	ARL16	RHD	ZGC:153405	SI:DKEY-95P16.2	SI:DKEY-269O24.6	RNF19A	PRCC
19	MLXIP	LYRM1	SI:CH211-221J21.3	SI:CH1073-159D7.4	SI:CH211-130H14.6	ZNF1065	CFL1	LAMA1
20	PTGFR	COPZ2	THEMIS2	LYRM1	SI:DKEY-16M19.4	SNAI2	SI:CH211-114N24.6	SI:CH73-36P18.5
21	SI:DKEY-261M9.15	NEU3.3	GRNAS	UHRF1	FAM78BB	LRRC8C	ITGA10	ZGC:153405
22	SI:CH73-368J24.14	SI:DKEY-148C4.3	SCOCB	HIST2H2AB	CDKN1BB	ZGC:111868	HNRNPA1B	SI:CH211-113A14.18
23	CHD8	SI:CH73-281F12.4	SI:CH211-276A17.5	ENSDART00000138896	TMED9	CTSC	PVALB9	LAMC1
24	PCNT	KLHL22	SI:CH211-160O17.6	ZNF1080	PDIA3	ZNF1064	SRSF1A	SI:DKEY-199M13.4
25	SI:CH211-113A14.24	SMARCD3A	SFT2D1	PPIG	SI:DKEY-5I16.2	SI:DKEY-43F9.4	RNASEP_NUC	SI:DKEYP-3B12.6
26	CX43	PAX7A	CSF1RB	ZGC:173552	SI:CH211-113A14.10	CA9	SLC17A7A	HSP90AA1.2
27	ITSN2B	H1M	MCHR2	H2AFVB	WEE2	ZGC:173705	HNRNPA0A	SI:DKEY-261M9.12
28	ZGC:110434	HSPE1	SI:CH211-178N15.1	HUWE1	SI:DKEY-23A13.15	SI:DKEY-156K2.3	SFPQ	FTR76
29	NANS	CABZ01001464.1	ZGC:154093	SI:CH211-113A14.18	SI:CH211-190K17.31	SI:DKEY-247I3.6	BAZ2A	MED13A

Table 1: Top 30 genes in batch modules



Cells are now much less separated by batch in the plot.

Remove batch effects from the original expression matrix

This batch effect removed expression matrix will be used later for spatial mapping.

read in the expression matrix used for running NMF

```
tbl.dir = "./NMF/Datasets/"
dataset = read.table(paste0(tbl.dir, "ALL_noBatchCorrection_var.txt"))
dataset_scl = read.csv("./NMF/Results/P2_use/tables/scaled_data.csv", row.names = 1)
```

Removed batch effect

This is done by subtracting the product of the batch matrices (portions of matrix G and C with the batch modules) from the original data matrix.

```
rmNMFbatch <- function(batch_modules, G, C, dataset_scl, dataset) {
    # multiply the matrices to calculate batch effect:
    batch_scl = as.matrix(G[, paste0("X", batch_modules)]) %*% as.matrix(C[batch_modules,
        ])

    # subtract it from the dataset used for running NMF
    batchRM_scl = dataset_scl - batch_scl  ##dataset_scl is the nonzero-median scaled dataset

    # correct for negative values
    batchRM_scl[batchRM_scl < 0] = 0

    # calculate non-zero median of each gene in the original dataset
    binaryData = dataset > 0  ##dataset is the original log dataset (before median scaling)
    dataset_2 = expm1(dataset)
    scl_fac = unlist(lapply(rownames(dataset_2), function(x) median(as.numeric(dataset_2[x,
        binaryData[x, ]]))))

    # calculate non-zero median of the scaled dataset
    binaryData = dataset_scl > 0
    dataset_2 = expm1(dataset_scl)
    scl_med = unlist(lapply(rownames(dataset_2), function(x) median(as.numeric(dataset_2[x,
        binaryData[x, ]]))))

    # calculate non-zero median of the scaled dataset with batch effect subtracted
    # binaryData=batchRM_scl>0 ##
    dataset_2 = expm1(batchRM_scl)
    batchRM_scl_med = unlist(lapply(rownames(dataset_2), function(x) median(as.numeric(dataset_2[x,
        binaryData[x, ]]))))

    med_fac = batchRM_scl_med/scl_med
    final_scl_fac = scl_fac * med_fac/scl_med
    # final_scl_fac=scl_fac/scl_med

    # the medians in the scaled dataset is adjusted to the median of the non-zero
    # medians transform the batch corrected dataset to its (more or less) original
    # scale
    # batchRM_unscl=log1p(sweep(expm1(batchRM_scl),1,scl_fac/median(scl_fac),'*'))
    batchRM_unscl = log1p(sweep(expm1(batchRM_scl), 1, final_scl_fac, "*"))
    return(batchRM_unscl)
}
K18_noBatch = rmNMFbatch(batch_modules18, nmf_K18$G, nmf_K18$C, dataset_scl, dataset)
```

Compare datasets before and after batch correction

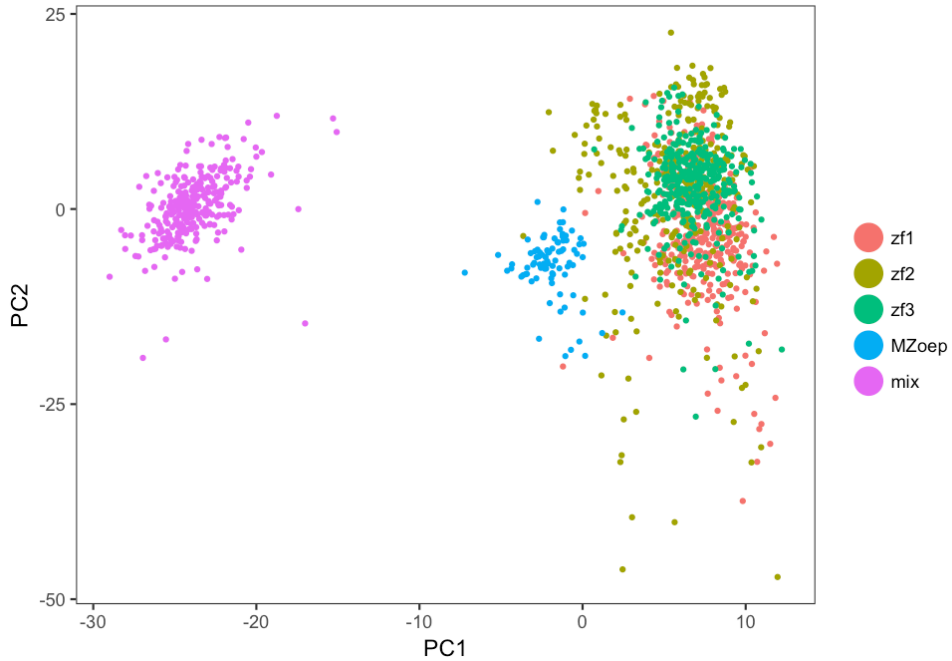
PCA plots are used to visualize transcriptomes in gene space before and after batch effect removal.

```
data_raw = new("seurat", raw.data = dataset)
K18_noBatch = new("seurat", raw.data = K18_noBatch)

data_raw = Setup(data_raw, project = "pre_batch_rm", min.cells = 1, names.field = 1,
    names.delim = "_", do.logNormalize = F, is.expr = 0.1, min.genes = 10)
K18_noBatch = Setup(K18_noBatch, project = "post_batch_rm", min.cells = 1, names.field = 1,
    names.delim = "_", do.logNormalize = F, is.expr = 0.1, min.genes = 10)

ident = data_raw@ident
levels(ident) <- c(levels(ident), "MZoep", "mix")
ident[grep("wt", ident)] = "mix"
ident[grep("oep", ident)] = "mix"
ident[grep("oep_p0", names(ident))] = "MZoep"
data_raw@ident = ident
```

```
K18_noBatch@ident = ident

data_raw = PCA(data_raw, pcs.print = 3, genes.print = 6, pc.genes = rownames(data_raw@data))
```

```
## [1] "PC1"
## [1] "SI:DKEY-153M14.1"  "ZGC:158463"        "SI:DKEYP-3B12.10"
## [4] "FXYD6"             "C1QB"              "SI:DKEY-108K21.12"
## [1] ""
## [1] "AGRP"           "LYRM1"          "DIABLOA"        "AGRP2"
## [5] "CABZ01084942.1" "HNRNPA0B"
## [1] ""
## [1] ""
## [1] "PC2"
## [1] "HS3ST3B1B"         "GRB7"              "ARID3A"
## [4] "RHD"               "SI:CH211-235I11.5" "SI:DKEY-186O21.1"
## [1] ""
## [1] "HMGB2A"           "HNRNPABB"         "SI:DKEY-151G10.6"
## [4] "HNRNPA0L"         "NPM1A"            "RAN"
## [1] ""
## [1] ""
## [1] "PC3"
## [1] "SI:DKEY-108K21.26"  "CX43"               "SMC1A"
## [4] "SI:CH211-113A14.8"  "MPL"                "SI:CH211-113A14.18"
## [1] ""
## [1] "SLC17A7A" "CYP4T8"   "SCN4BB"   "PVALB9"   "NTF3"     "PRAM1"
## [1] ""
## [1] ""
```

```
PCAPlot(data_raw, 1, 2, pt.size = 0.75)
```

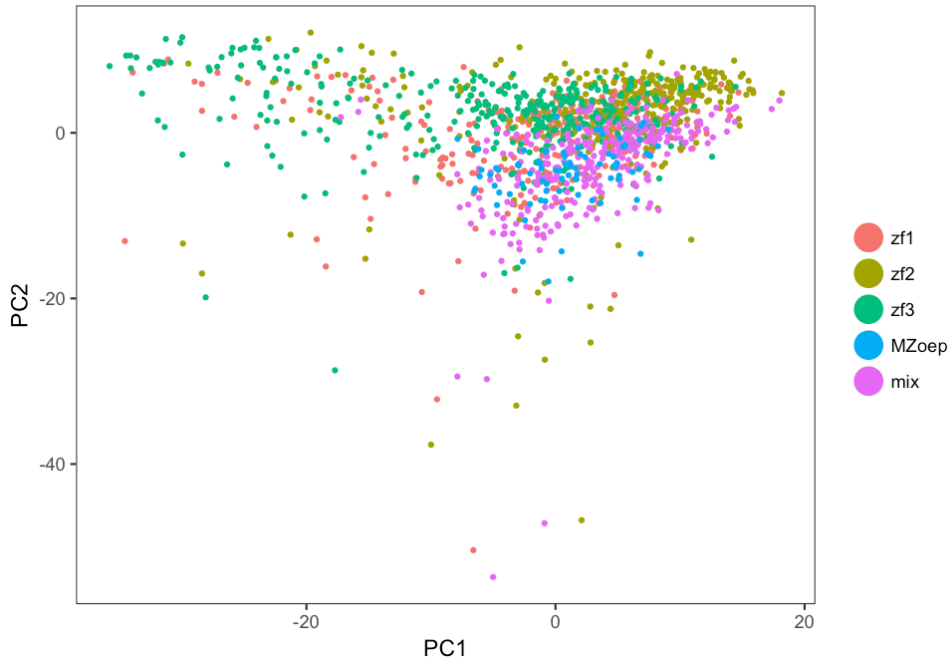



```
K18_noBatch = PCA(K18_noBatch, pcs.print = 3, genes.print = 6, pc.genes = rownames(K18_noBatch@data))
```

```
## [1] "PC1"
## [1] "OSR1"     "KIRREL3L" "LYRM1"    "SNAI1A"   "FSCN1A"   "EFNB2B"
## [1] ""
## [1] "SOX3"             "SI:DKEY-11O15.10" "FAM212AA"
## [4] "SI:CH211-170D8.2" "SOX19A"           "CXCR4B"
## [1] ""
## [1] ""
## [1] "PC2"
## [1] "KRT4"     "CLDNE"    "KRT5"     "KRT8"     "KRT92"    "MID1IP1A"
```

```
## [1] ""
## [1] "SI:DKEY-108K21.23" "NOTUM1A"           "RND1L"
## [4] "TOP1"              "DNTT"              "CABZ01084942.1"
## [1] ""
## [1] ""
## [1] "PC3"
## [1] "DIABLOA"         "FOXA3"           "AGRP"            "LYRM1"
## [5] "CABZ01084942.1" "AGRP2"
## [1] ""
## [1] "ALDOB"             "SOX3"              "SOX2"
## [4] "SI:CH211-222L21.1" "ZIC2B"             "SOX19A"
## [1] ""
## [1] ""
```

```
PCAPlot(K18_noBatch, 1, 2, pt.size = 0.75)
```



Save the dataset with batch effect removed for spatial mapping with Seurat

```
save.dir = "./Datasets/"
write.table(K18_noBatch@data, file = paste0(save.dir, "Batch_corrected_var.txt"),
    quote = FALSE, sep = "\t")
```

Clustering analysis of cells based on non-batch gene modules

First, Scale matrix C and assign names to modules

The C matrix (non-batch modules by cells) is scaled such that each row has the same maximum value before clustering. Modules are assigned names according to knowledge about their top ranked genes.

```
maxScl <- function(df, dir = "row", max_value = NULL, log_space = F) {
    if (dir == "row") {
        dir = 1
    } else if (dir == "col") {
        dir = 2
    } else {
        print("dir must be 'row' or 'col'.")
        return
    }
```

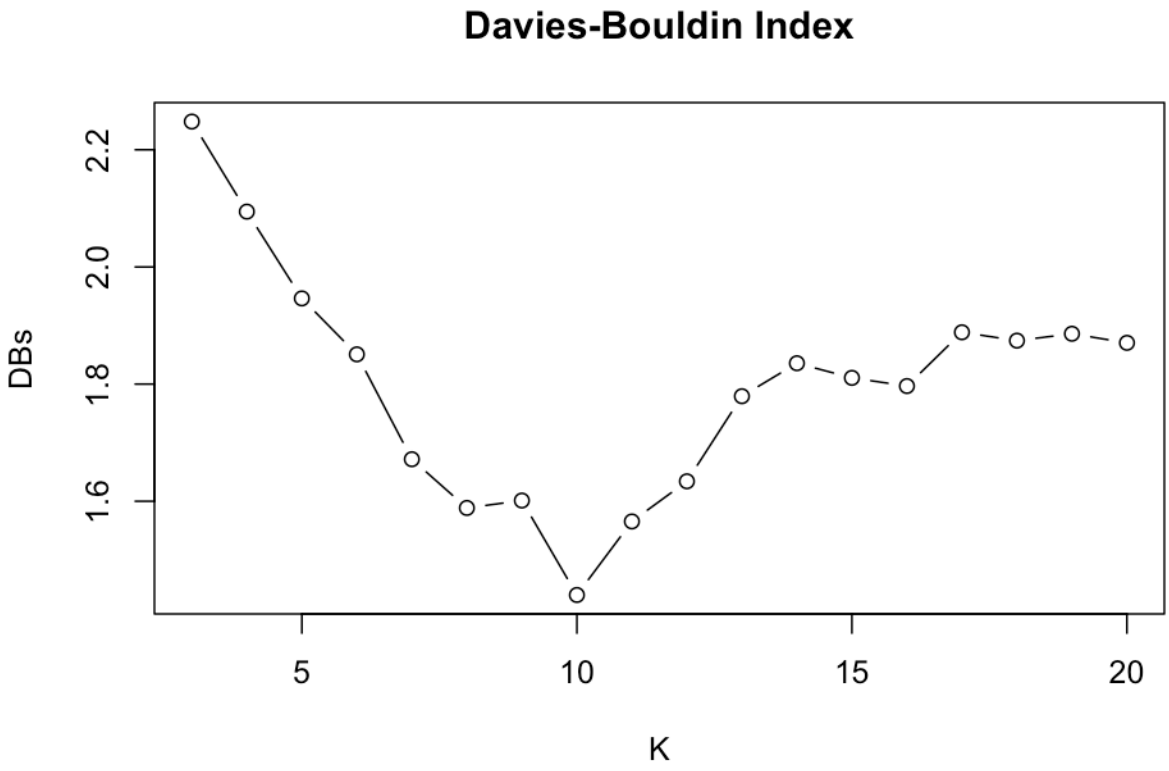
```
    if (is.null(max_value)) {
        max_value = median(apply(df, dir, max))
    }
    if (log_space) {
        df = expm1(df)
        max_value = expm1(max_value)
    }
    df_scl = sweep(df, dir, apply(df, dir, max), "/")
    df_scl = df_scl * max_value
    if (log_space) {
        df_scl = log1p(df_scl)
    }
    return(df_scl)
}

scld_C = maxScl(nmf_K18$C)
rownames(scld_C) = c("0", "1", "2", "Marginal", "EVL", "5", "6", "7", "Marginal Dorsal",
    "Dorsal", "10", "Apoptotic-like", "YSL", "13", "PGC", "Ventral Animal", "Dorsal Animal",
    "Ventral")
```

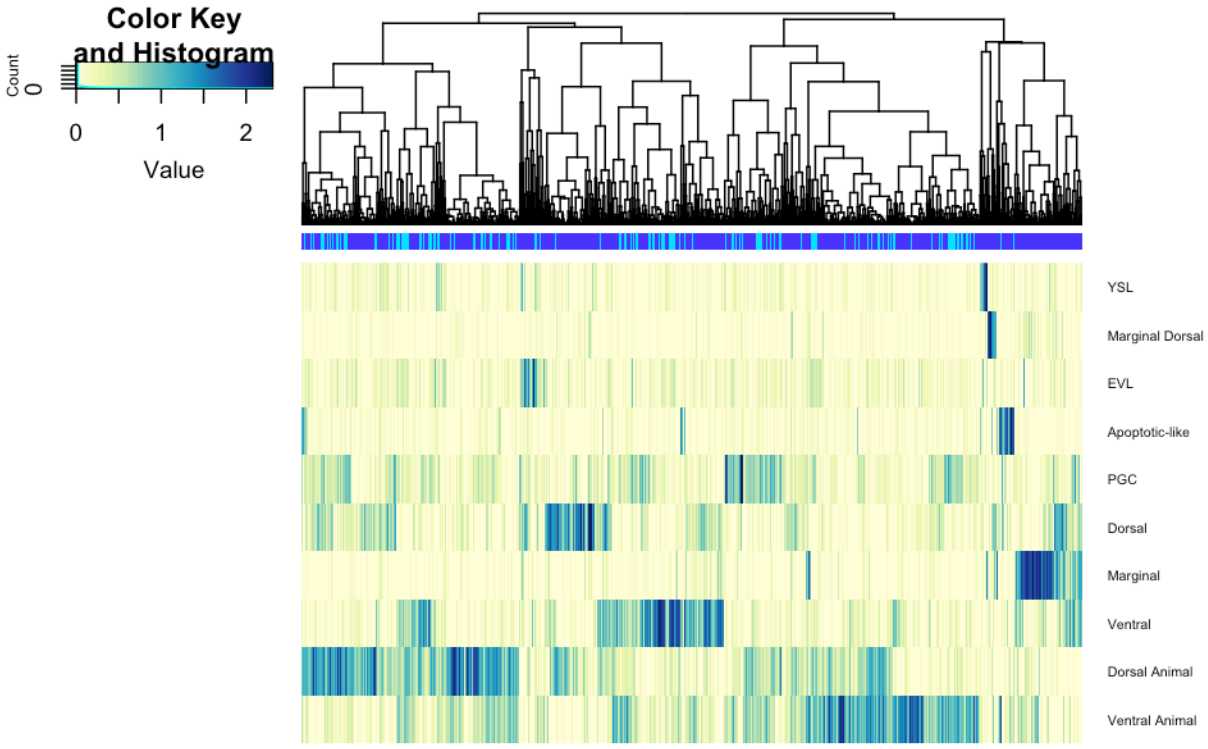
Apply hierarchical clustering

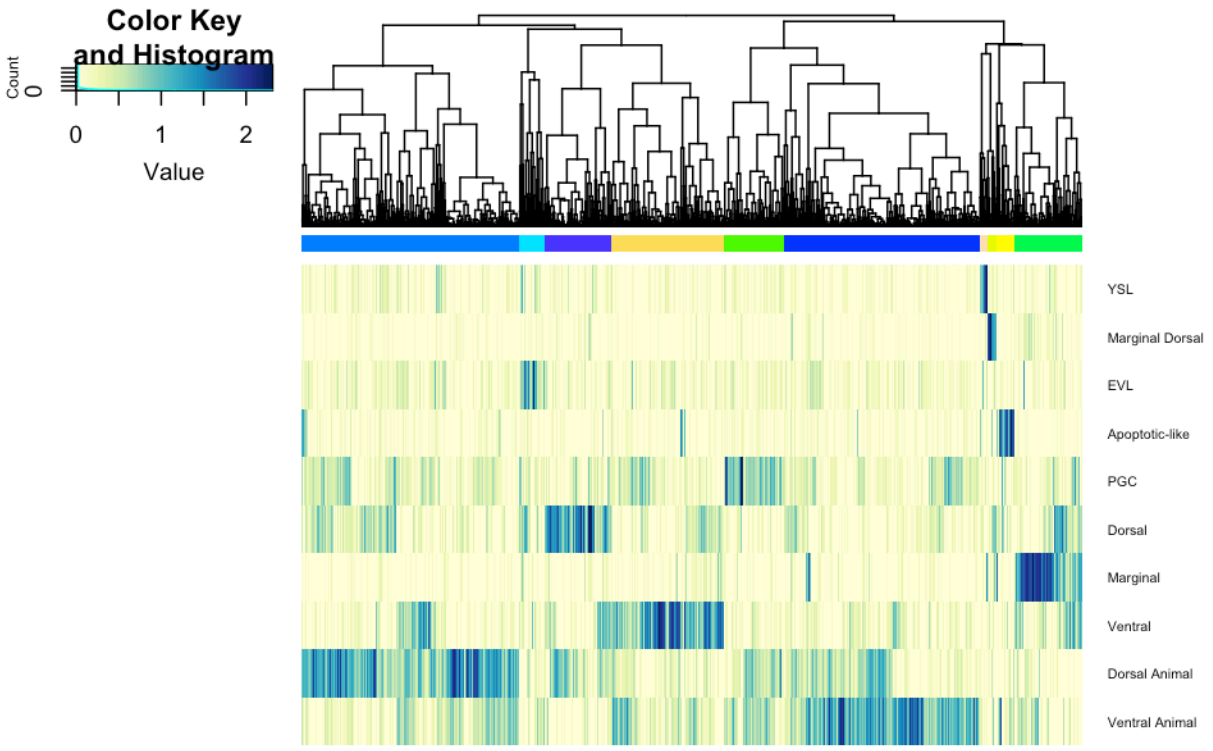
Davies-Bouldin index is used to determin the optimal number of clusters (as the number that gives the lowest DB index). Clustering result is shown as heatmaps, with color bars on top indicating genotype (dark blue = wt; light blue = MZoep) or cluster membership.

```
library(gplots)
library(RColorBrewer)
library(clusterSim)
cluster_map <- function(scld_C, genos = NULL, group = c("cluster", "geno"), group_colors = NULL,
    den_cut = 0.75, metric = c("cor", "dist"), method = "complete", DB = F) {
    if (metric == "cor") {
        stds = apply(scld_C, 2, sd)
        zero_var = which(stds == 0)
        if (length(zero_var) > 0) {
            print(paste("removing ", length(zero_var), "zero-variance cell(s) from dataframe:"))
            print(colnames(scld_C)[zero_var])
            scld_C = scld_C[, -zero_var]
        }
        hc <- hclust(as.dist(1 - cor(as.matrix(scld_C))), method = method)
    } else if (metric == "dist") {
        hc <- hclust(dist(as.matrix(t(scld_C)), method = "euclidean"), method = method)
    }
    if ("cluster" %in% group) {
        if (den_cut < 1) {
            mycl <- cutree(hc, h = max(hc$height) * den_cut)
        } else {
            mycl <- cutree(hc, k = den_cut)
        }
        if (is.null(group_colors)) {
            mycolhc <- topo.colors(length(unique(mycl)))
            mycolhc <- mycolhc[as.vector(mycl)]
        } else {
            mycolhc <- group_colors[as.vector(mycl)]
        }
        cluster = mycolhc
    }
    if ("geno" %in% group) {
        geno_code = vector("numeric", length = dim(scld_C)[2])
        geno_code = geno_code * 0 + 1
        i = 1
        for (geno in genos) {
            geno_code[grep(geno, colnames(scld_C))] = i * 2
            i = i + 1
```

```
        }
        if (is.null(group_colors)) {
            mycolgeno <- topo.colors(2 * length(genos))
            mycolgeno <- mycolgeno[geno_code]
        } else {
            mycolgeno <- group_colors[geno_code]
        }
        geno = mycolgeno
    }
    hmcol <- colorRampPalette(brewer.pal(9, "YlGnBu"))(100)
    if (DB) {
        DBs = c()
        for (i in c(3:20)) {
            mycl <- cutree(hc, k = i)
            if (metric == "cor") {
                DB = index.DB(t(scld_C), mycl, dist(1 - cor(as.matrix(scld_C))),
                  centrotypes = "centroids")
            } else if (metric == "dist") {
                DB = index.DB(t(scld_C), mycl, dist(as.matrix(t(scld_C)), method = "euclidean"),
                  centrotypes = "centroids")
            }
            DBs = c(DBs, DB$DB)
        }
        plot(c(3:20), DBs, type = "b", main = "Davies-Bouldin Index", xlab = "K")
    }
    for (grp in group) {
        heatmap.2(as.matrix(scld_C), Colv = as.dendrogram(hc), ColSideColors = get(grp),
            col = hmcol, dendrogram = "column", sepwidth = 0, trace = "none", labCol = "",
            cexRow = 0.65)
    }
    return(mycl)
}

cluster_info = cluster_map(as.matrix(scld_C[setdiff(rownames(scld_C), batch_modules18),
    ]), group = c("geno", "cluster"), genos = c("oep"), metric = "cor", den_cut = 10,
    DB = T)
```


Davies-Bouldin Index
DBs
1.6
1.8
2.0
2.2
5
10
15
20
K

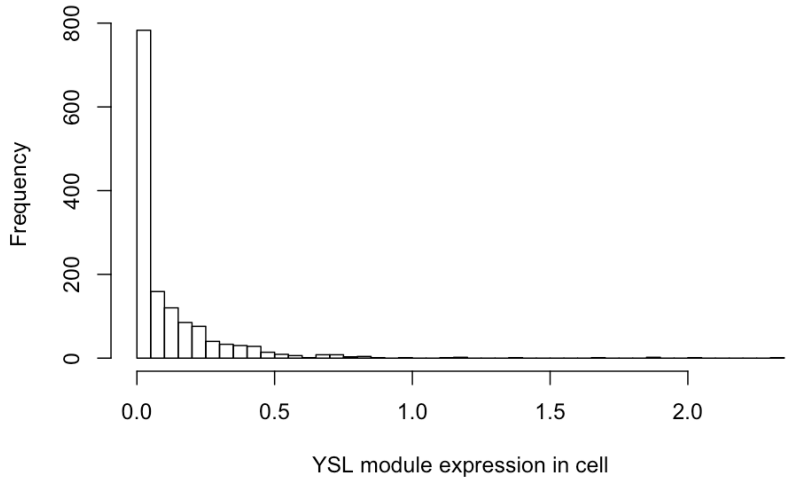

Color Key and Histogram
Count
0
0
1
2
Value
YSL
Marginal Dorsal
EVL
Apoptotic-like
PGC
Dorsal
Marginal
Ventral
Dorsal Animal
Ventral Animal



Repeat cluster without cells expressing high levels of YSL module

YSL is intensinally removed in our sample collecting procedure. The YSL module detected is likely resulted from incomplete yolk removal or contamination from YSL.

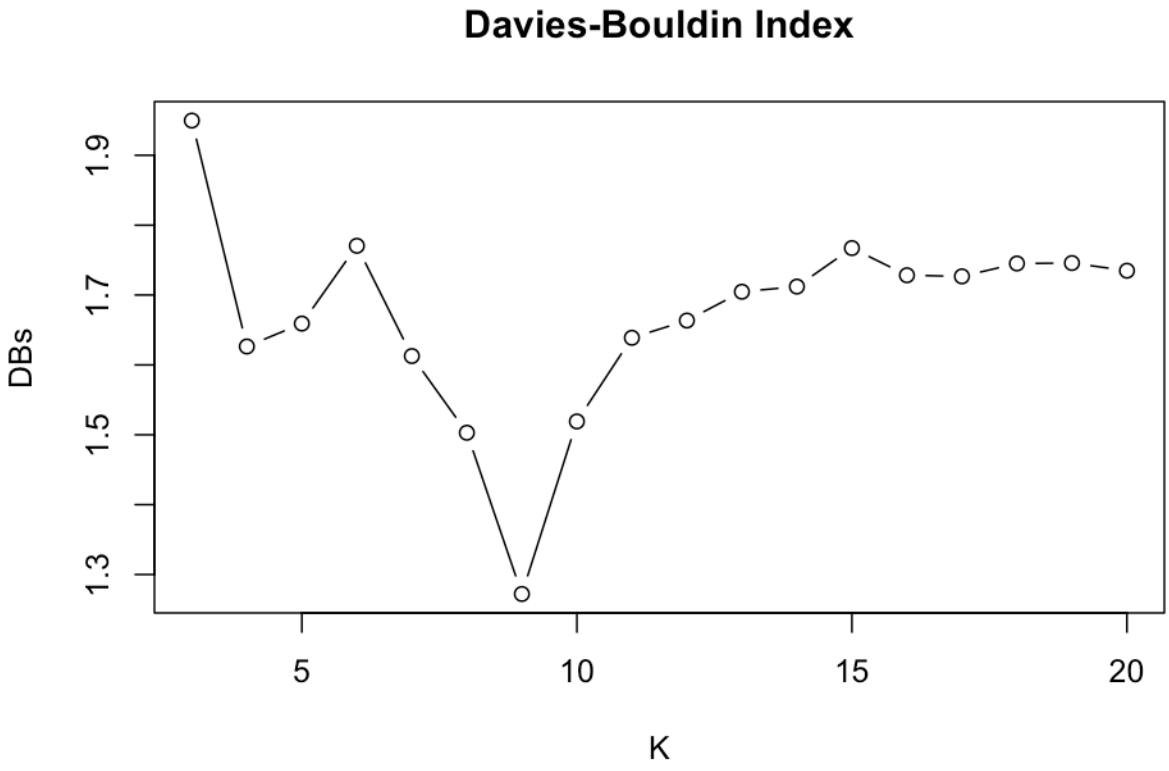
```
hist(as.numeric(scld_C["YSL", ]), breaks = 50, main = "", xlab = "YSL module expression in cell")
```

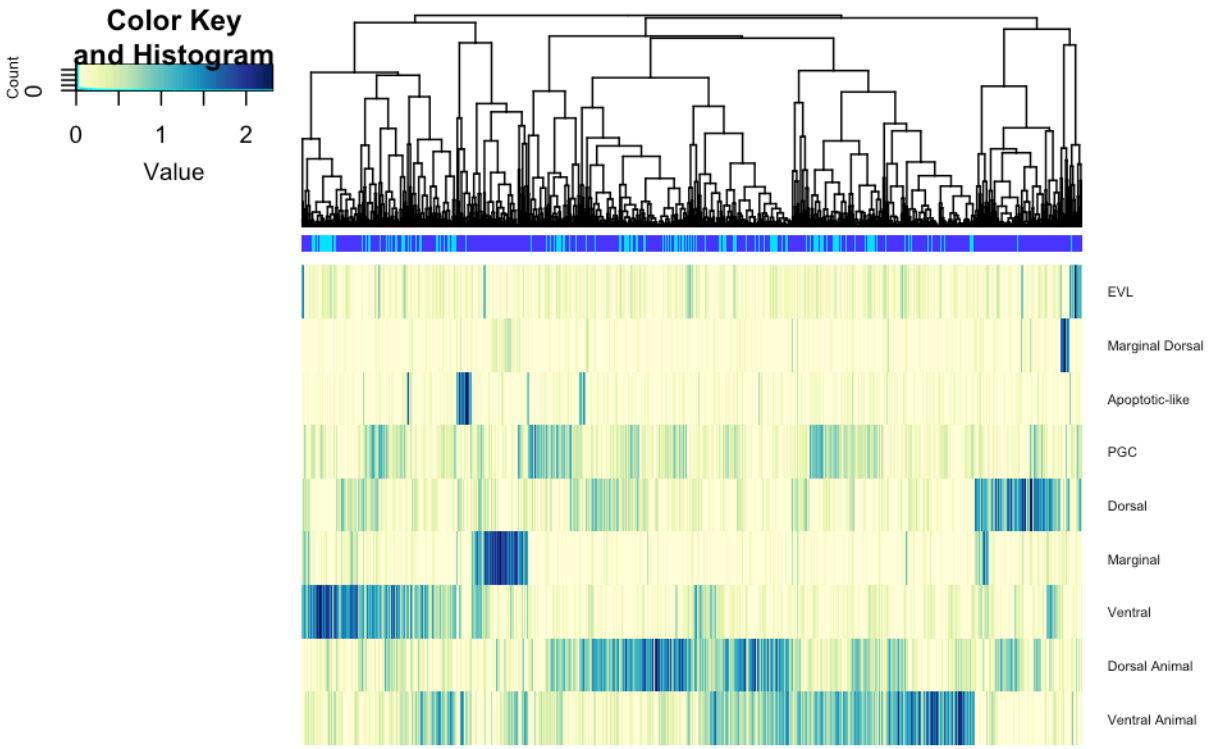


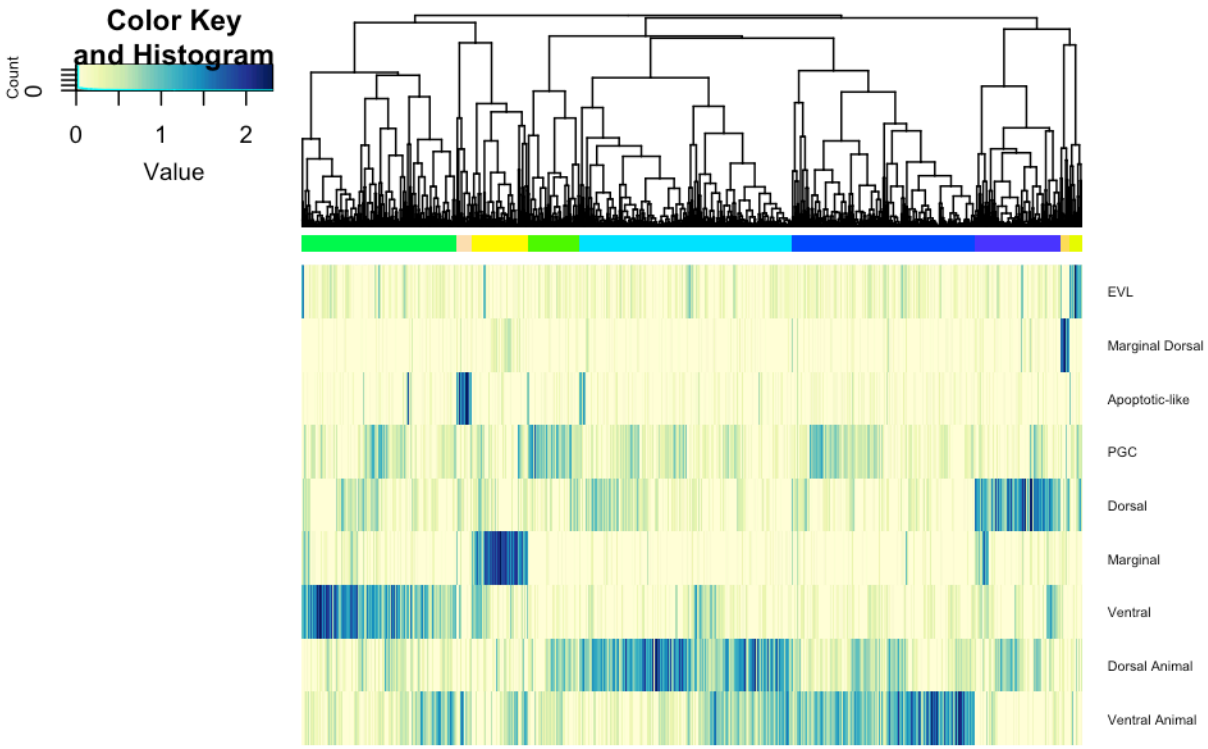

```
# length(which(scld_C['YSL',]>1)) #=9
C_noYSL = scld_C[, colnames(scld_C)[which(scld_C["YSL", ] < 0.7)]]
```

Again, we use Davies-Bouldin index to determin the optimal number of clusters, and show clustering result as heatmaps, with color bars on top indicating genotype (dark blue = wt; light blue = MZoep) or cluster membership.

```
cluster_info = cluster_map(as.matrix(C_noYSL[setdiff(rownames(C_noYSL), c("YSL",
    batch_modules18)), ]), group = c("geno", "cluster"), genos = c("oep"), metric = "cor",
    den_cut = 9, DB = T)
```




Color Key
and Histogram
Count
0
0
1
2
Value
EVL
Marginal Dorsal
Apoptotic-like
PGC
Dorsal
Marginal
Ventral
Dorsal Animal
Ventral Animal


Color Key
and Histogram
Count
0
0
1
2
Value
EVL
Marginal Dorsal
Apoptotic-like
PGC
Dorsal
Marginal
Ventral
Dorsal Animal
Ventral Animal